Summary

This document covers the reasoning and challenges faced with the implementation, how challenges were overcome and what motivated the developer to complete all the objectives within the limited time. This document also covers the architectural diagram and the results achieved. Additionally, this document contains links to the resources used, as well as the architectural diagram.

The detailed instructions to install the implemented solution into an AWS account and test the solution is explained in the User manual documentation in the documentations folder.

To test the application, make a call to **+441143921064**

Table of Contents

[**Thought Process** 2](#_Toc91013258)

[**Reasoning** 2](#_Toc91013259)

[**Challenges** 3](#_Toc91013260)

[**Best Practices** 3](#_Toc91013261)

[**Shortcuts and things that would be improved with more time** 3](#_Toc91013262)

[**Shortcuts** 3](#_Toc91013263)

[**Improvements that would be added with time** 3](#_Toc91013264)

[**Appendix A: Architectural Diagram** 4](#_Toc91013265)

[**Appendix B: Execution & Result Screenshots** 5](#_Toc91013266)

# **Thought Process**

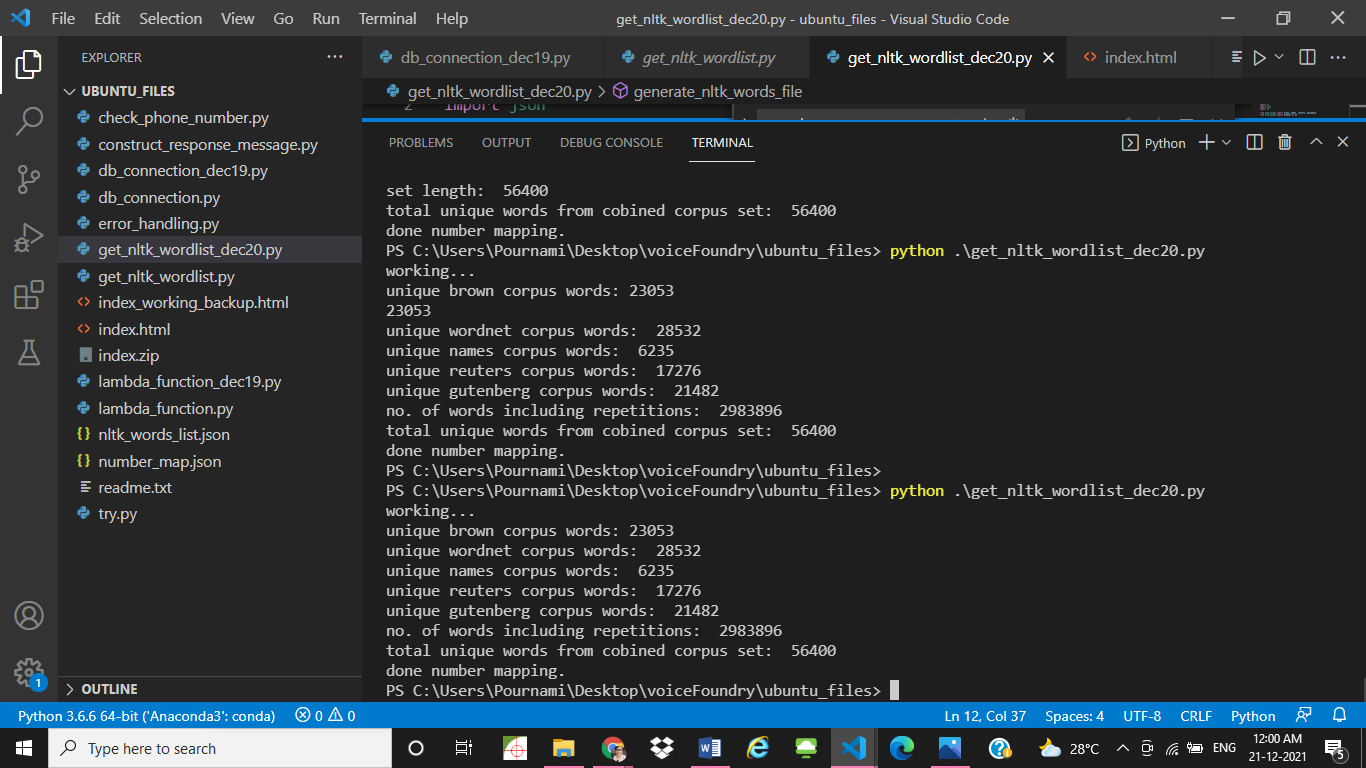
The major thought process around the assignment on vanity number generator was the logic to generate vanity Numbers by exploring the telephone number format followed in UK. The next questionable thoughts were implementing Amazon connect, creating web app to display data from DynamoDB and getting deployable package using Cloudformation/CDK/SAM, most of which being unfamiliar AWS consoles, felt challenging and at the same time motivating to complete within the one week time span.

## **Reasoning**

The first aim was to create a personal AWS account, explore Amazon connect and establish a working connection by claiming a phone number as voiceFoundry majorly focuses on deploying applications on Amazon Connect. Next aim was to explore available formats for UK based phone numbers and find the common format to find the country codes and area codes, being a fresh migrant to UK.

The next main thought was on getting the best collection of words to generate vanity Numbers. Referred <https://github.com/dwyl/english-words> which is a dictionary of English words, but was not a standard one to use in production environments. Initially explored the NLTK word corpus, used words having number of letters in between 2 and 7. The results were not as good as expected with the word corpus with the filters applied for word selection from the corpus.

NLTK brown corpus gave better results compared to word corpus, but the total number of unique words in the word number map generated was less than 30,000. Explored the NLTK corpus, which is a standard corpora to use for commercial production purposes. Used wordnet, brown, Gutenberg, reuters and names corpus of NLTK. All words were fetched, preprocessed to remove duplicates, words with Unicode characters, filtered out words with only alphabets with length between 2 and 7, making the total word count 56400 by removing the recurring words from combined corpora of 2983896 words.



The algorithm was designed to keep the longest vanity word with the highest rank as it will be most memorable compared to combinations of smaller length words, with numbers in between.

## **Challenges**

The major challenges revolved around the exploration of amazon connect and amazon amplify consoles, which being new exploratory areas. Exploring Amazon connect was interesting and fun filled to see the different blocks available to create the contact flow and make it more interactive to the user. But however the connection from Amazon connect to lambda took developers precious time. The troubleshooting methods and explorations in tutorials finally guided the developer to set the required permissions for amazon connect to send invocation to lambda.

Web App development from AWS was clueless unless developer came across the fruitful documentation on building webapp using Amazon Amplify. All thanks to the well explained documentation support team of Amazon in detailing the minute steps which makes it easier for even a beginner to become an expert in all AWS consoles. The main challenge which took developers another couple of fruitful hours was by using a filename other than index.html for amplify console. Getting ‘XML does not seem to have any style information associated with it’ took a long time to troubleshooting by trial and error method to finally realize the silly mistakes that can cost hours of a developers life.

Another challenge was working with personal system having limited specifications and was time-consuming as the performance was low. However the patience and determination helped in completing all the objectives of this assignment within the provided time along with busy work life and family life.

More than overcoming challenges, each of the struggles were best lessons of AWS the developer ever learned.

# **Best Practices**

Adding error handling to the lambda always made the errors transparent as SNS notifications reaches the mail subscribed as soon as an exception occurs in the lambda invocation. Error messages were also sent to DLQ to be looked up later.

Added environment variables for ARN value of SNS, Queue URL of SQS and part of dynamo db table. Also in lambda for setting permissions, added account Id in environment variables as a standard approach.

# **Shortcuts and things that would be improved with more time**

## **Shortcuts**

Use of reference number word mapping file than using real-time vanity word generation.

Explored most of the standard NLTK text corpus for best collection of words. But with time, even the corpus would have been made bigger by exploring more available resources.

Very few unit testing done due to limitations in time. More unit test cases needs to be tested for better performance assurance.

## **Improvements that would be added with time**

With more time, the changes that would be added are:

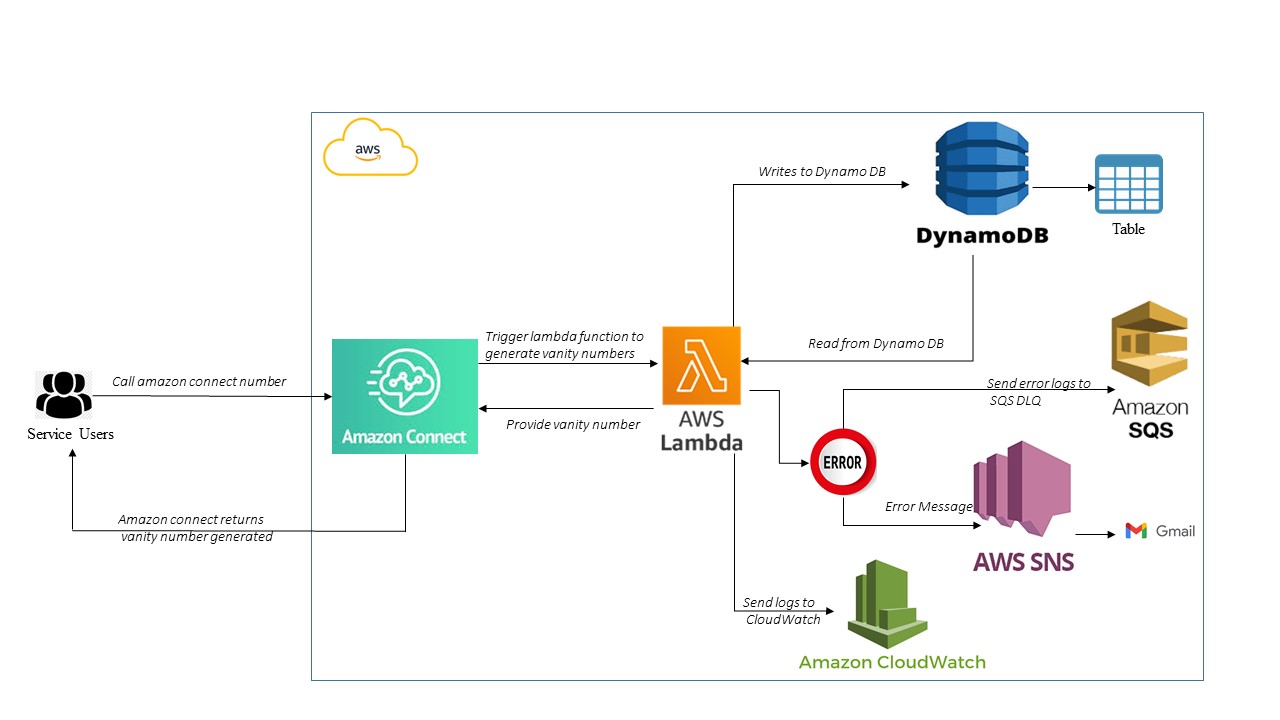
-multiple user queues and routing in amazon connect.

-Use amazon pinpoint to send the top3 vanity number generated to service users phone number.

-Real-time generation of vanity words rather than using a number-word map file.

-More try-except blocks to handle all the possible errors.

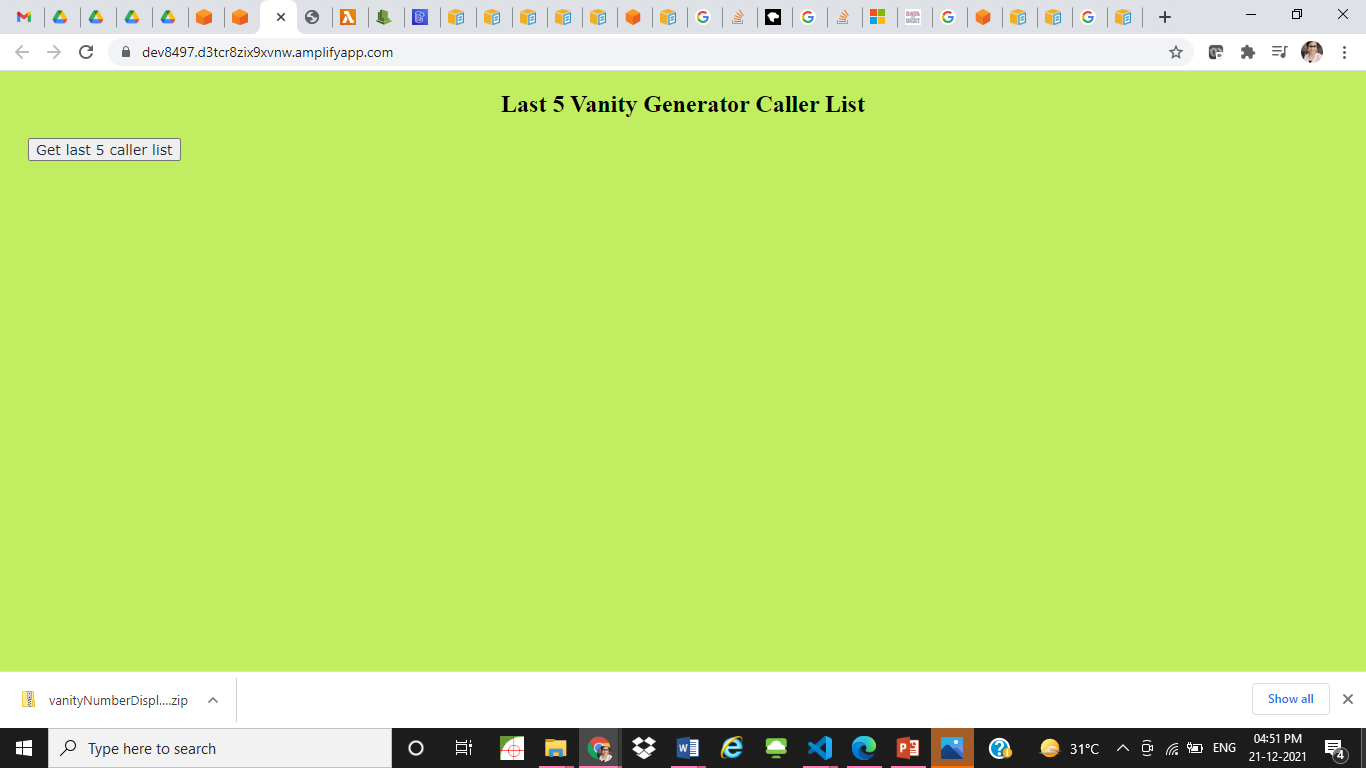
# **Appendix A: Architectural Diagram**

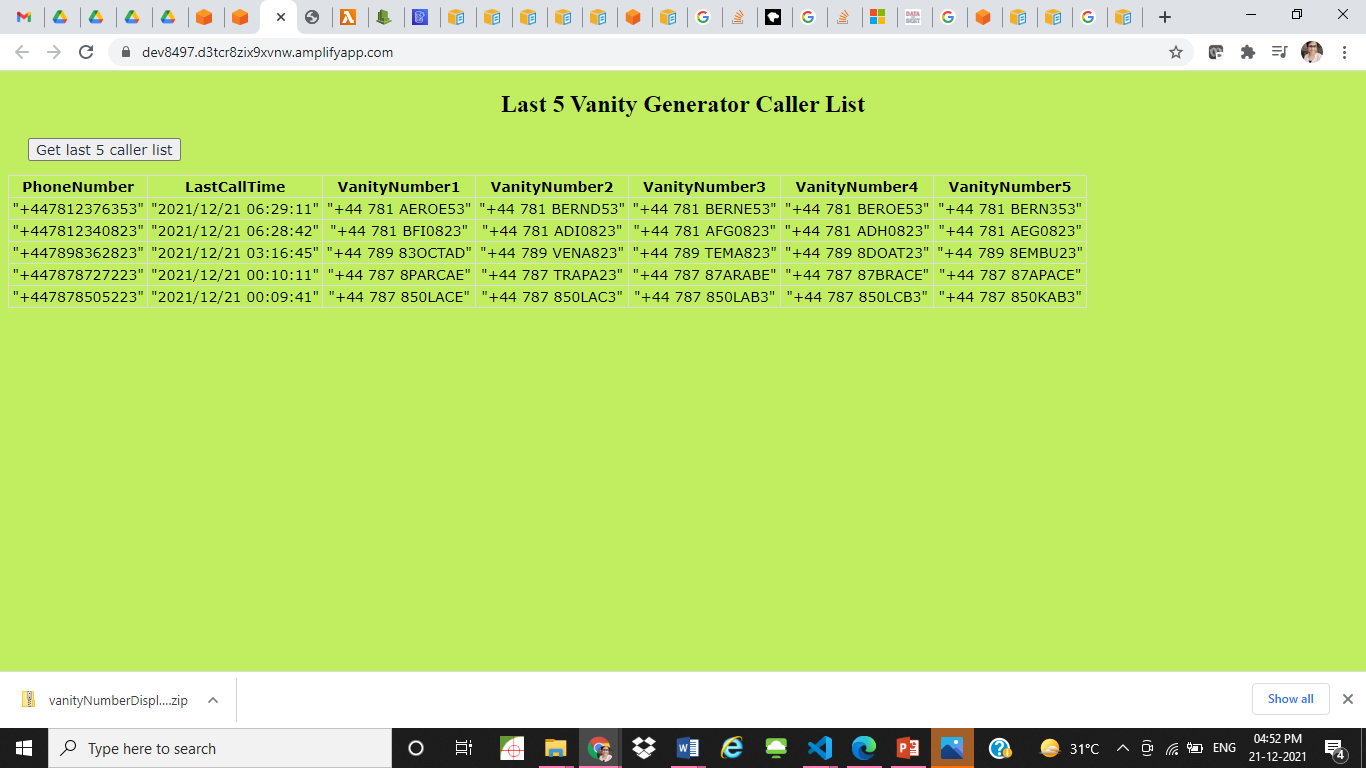


Architectural Diagram for the Vanity Phone Number Generator

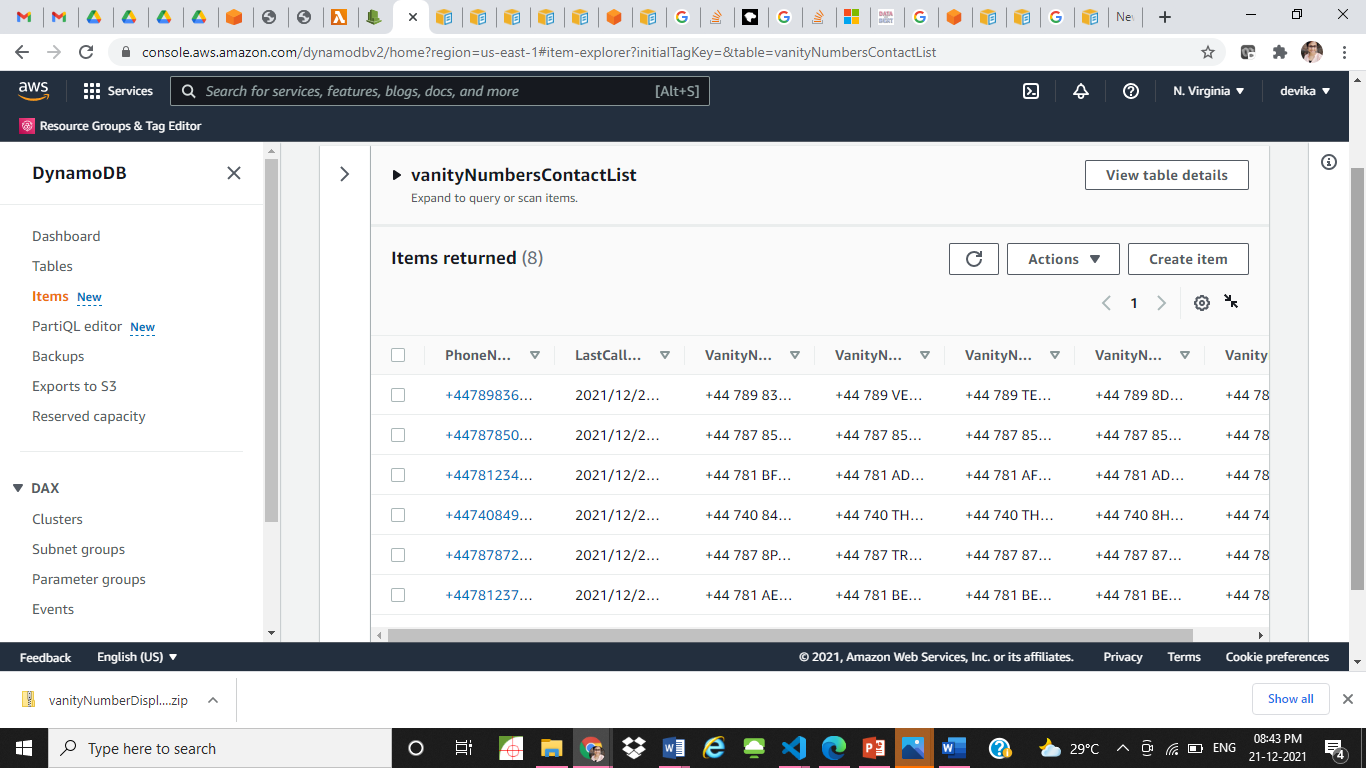
# **Appendix B: Execution & Result Screenshots**

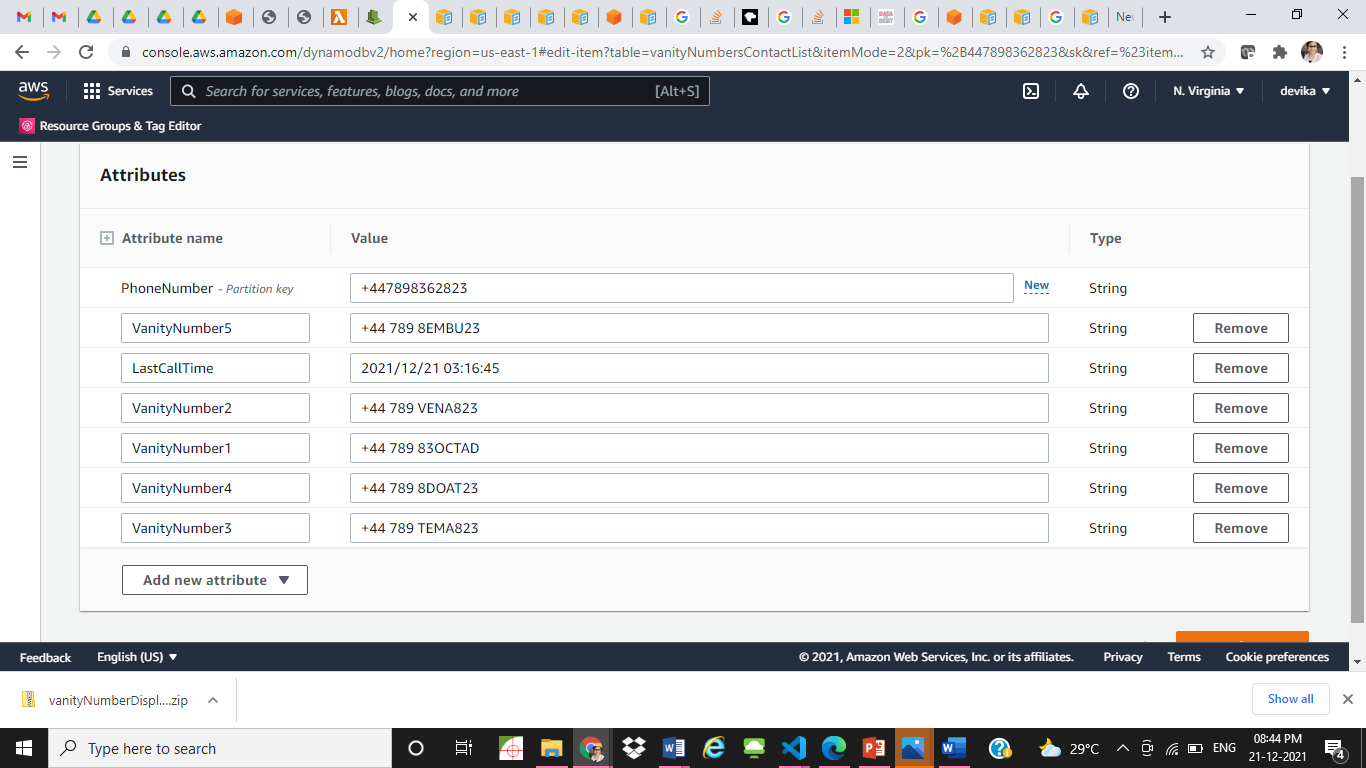
Web App



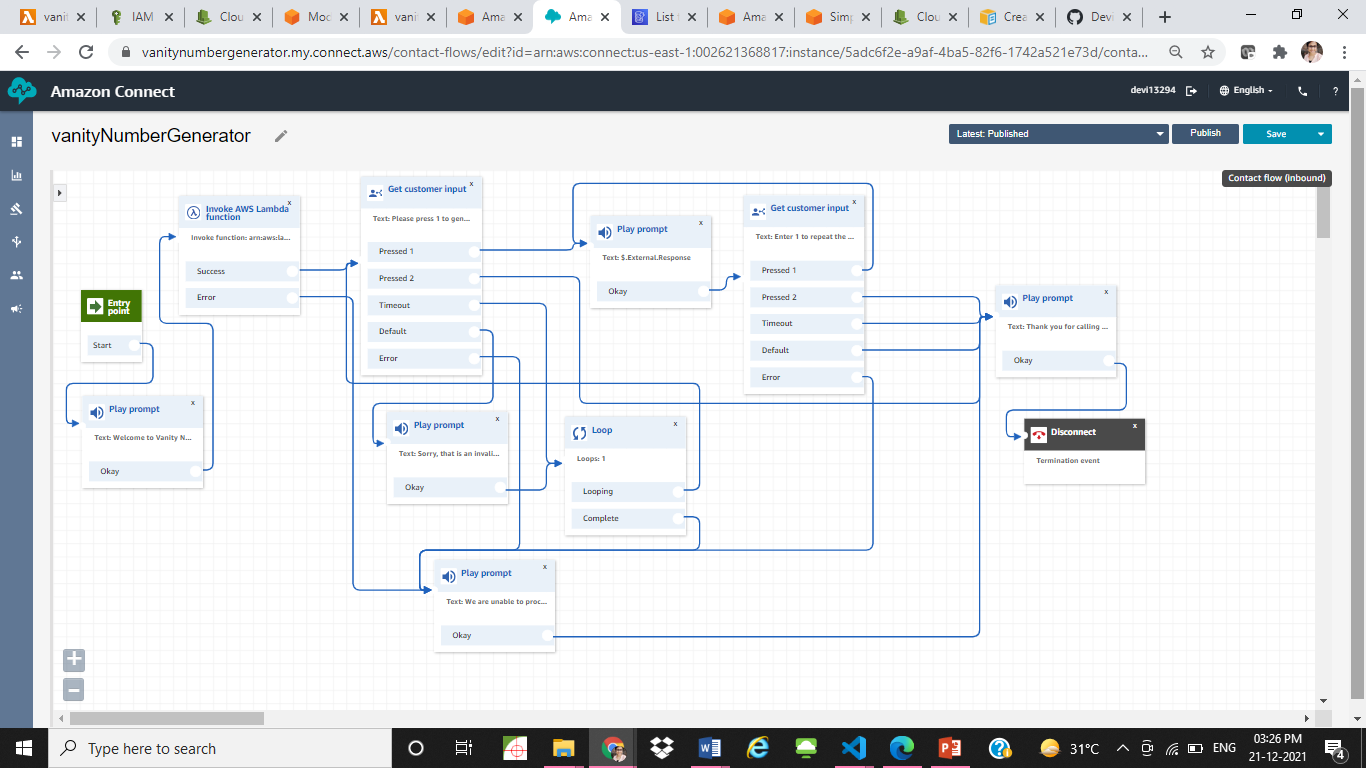


DynamoDB table

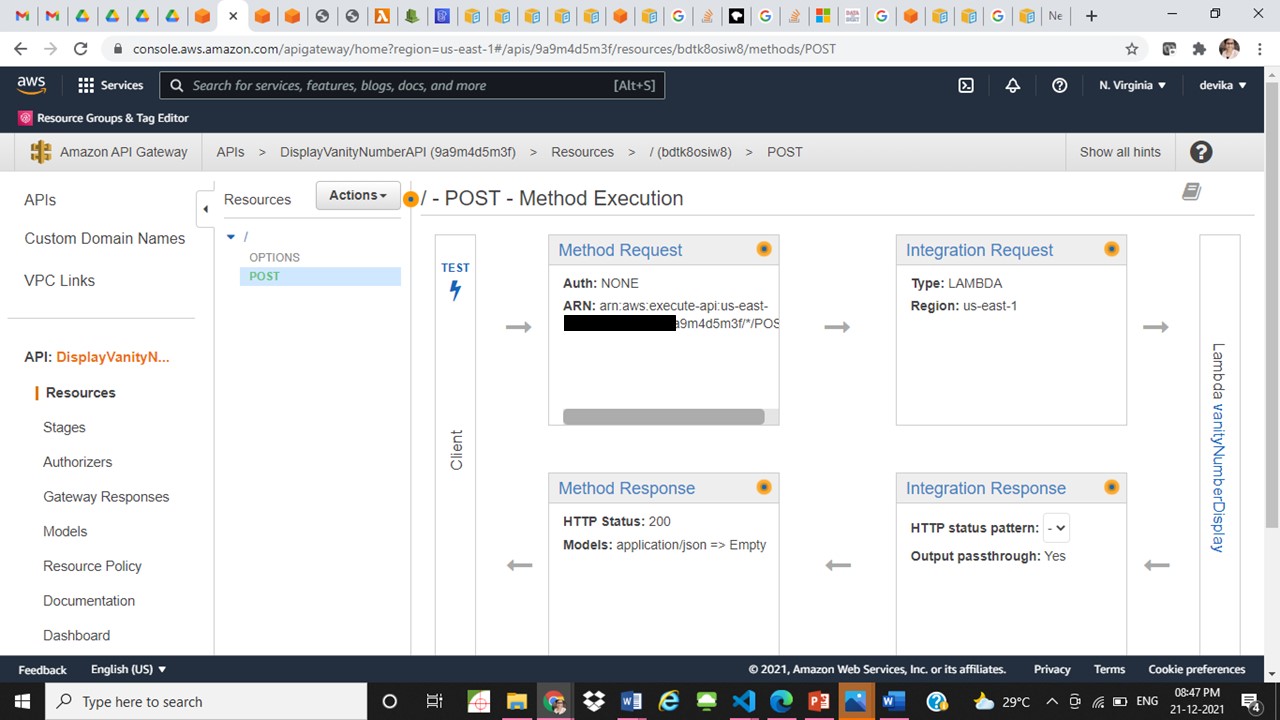




Amazon Connect Contact flow



API Gateway for web app



Amazon Amplify Console

