SMART EMAIL ASSISTANT

A PROJECT WORK REPORT

Submitted by

KAVYA. A (1601107)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

SRI RAMAKRISHNA ENGINEERING COLLEGE
COIMBATORE – 641 022

ANNA UNIVERSITY: CHENNAI 600 025

JULY 2020

SRI RAMAKRISHNA ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

Department of Computer Science and Engineering

PROJECT WORK - JULY 2020

This is to certify that the project entitled

SMART EMAIL ASSISTANT

is the bonafide record of project work done by

KAVYA A

Register No: 1601107

who carried out the project work under my supervision, certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

of B.E Computer Science and Engineering, during the year 2016-2020.

Dr. A. Grace Selvarani
Head of the Department

Mr. R. Kanagaraj
Guide

Submitted for the Project Viva-Voce examination held on 14.07.2020

Internal Examiner	Subject Expert	Subject Expert
C.Padmavathy AP (Sl.Gr)	Dr. R Anuradha Associate Professor	Dr. A. Grace Selvarani Professor & HoD

ACKNOWLEDGEMENT

We express our gratitude to **Sri. D. LAKSHMINARAYANASWAMY,** Managing Trustee, SNR Sons Charitable Trust, Coimbatore for providing excellent facilities to carry out our project.

We express our deepest gratitude to our Principal, **Dr. N. R. ALAMELU., Ph.D.,** for her valuable guidance and blessings.

We are indebted to our Head of the Department, **Dr. A. Grace Selvarani, Ph.D.,** Department of Computer Science and Engineering who modelled us both technically and morally for achieving great success in life.

We express our thanks to our Project Coordinator, **Mr.R.Kanagaraj,** Assistant Professor (Sl. Grade) Department of Computer Science and Engineering for her great inspiration.

Words are inadequate to offer thanks to our respected guide. We wish to express our sincere thanks to **Dr. R. Anuradha,** Associate Professor, Department of Computer Science and Engineering, who gives constant encouragement and support throughout this project work and who makes this project a successful one.

We also thank all the staff members and technicians of our Department for their help in making this project a successful one.

VELVET BRICK
SELLING STRATEGIES

Date: Monday, June 22, 2020

SUB: INTERNSHIP COMPLETION LETTER

We are glad to inform you that **Kavya Ayyappan** of final year students of B.E.(CSE) from **Sri Ramakrishna Engineering College** has successfully completed her internship at **Velvet Brick Selling Strategies** from **Dec 2019** to **May 2020** on a project titled "**SMART EMAIL ASSISTANT"**.

During this internship, Kavya worked on various software product development and successfully met the objectives that were set at the beginning of the project. We found her extremely inquisitive and hard working with a self-motivated attitude to learn new things. Her performance exceeded expectations and was able to complete the project successfully on time.

We wish her all the best for her future endeavors and look forward to work with her again in future.

Warm Regards,
Bryan Siever
[FOUNDER]

+1 (669)699-8323
bryan.siever@ gmail.com
www.24thwave.com
18 Arroyo Vista Drive, Goleta, CA

ABSTRACT

In recent years, artificial intelligence (A.I.) has become an emerging trend in different fields: science, business, medicine, automotive, and education. A.I. has also reached marketing.

The purpose of this project is to collect emails from customers and provide email insights to the marketers. With the recent shift in customers using digital platform to communicate with businesses, it is a challenging task for a business representative to go through all the emails, prioritize them and take necessary actions. Increased email communications not only add complexity, but also reduces the quality of business correspondence to a customer's enquiry. Some of the customer communication challenges faced by a business, can be overcome with the help of Smart Email Assistant by parsing through the email and extracting interesting objects from an email.

With Smart Email Assistant provides marketers to built better understand their customers by closely monitoring customer email exchanges and by providing email insight dashboard to marketers. Email insight includes keyword selection and categorization, text summarization, sentiment analysis, emotional reaction, entity extraction and taxonomy classification. Smart Email Assistant also provides a context sentence extraction capability for a set of keywords administered by a marketer**.**

User emails are stored and processed by MailParser inboxes, via a multi-step process which includes, storing, parsing, creating data objects, and invoking a pre-configured webhook dispatch. Parsed email is then sent to AWS Lambda through API Gateway by making a HTTP GET method request, and in response generate insights.

சுருக்கம்

சமீபத்திய ஆண்டுகளில், செயற்கை நுண்ணறிவு பல்வேறு துறைகளில் வளர்ந்து வரும் போக்காக மாறியுள்ளது:அறிவியல், வணிகம், மருத்துவம், வாகன மற்றும் கல்வி. ஏ.ஐ. சந்தைப்படுத்துதலையும் அடைந்துள்ளது.

இந்த திட்டத்தின் நோக்கம் வாடிக்கையாளர்களிடமிருந்து மின்னஞ்சல்களை சேகரித்து சந்தைப்படுத்துபவர்களுக்கு மின்னஞ்சல் நுண்ணறிவுகளை வழங்குவதாகும்.வணிகங்களுடன் தொடர்புகொள்வதற்கு டிஜிட்டல் தளத்தைப் பயன்படுத்தும் வாடிக்கையாளர்களின் சமீபத்திய மாற்றத்துடன், ஒரு வணிக பிரதிநிதி அனைத்து மின்னஞ்சல்களிலும் சென்று, அவர்களுக்கு முன்னுரிமை அளித்து, தேவையான நடவடிக்கைகளை எடுப்பது ஒரு சவாலான பணியாகும். அதிகரித்த மின்னஞ்சல் தகவல்தொடர்புகள் சிக்கலைச் சேர்ப்பது மட்டுமல்லாமல், வாடிக்கையாளரின் விசாரணைக்கு வணிக கடிதங்களின் தரத்தையும் குறைக்கிறது. ஒரு வணிகத்தால் எதிர்கொள்ளும் சில வாடிக்கையாளர் தொடர்பு சவால்களை ஸ்மார்ட் மின்னஞ்சல் உதவியாளரின் உதவியுடன் மின்னஞ்சல் மூலம் பாகுபடுத்தி, மின்னஞ்சலில் இருந்து சுவாரஸ்யமான பொருட்களைப் பிரித்தெடுப்பதன் மூலம் சமாளிக்க முடியும்.

வாடிக்கையாளர் மின்னஞ்சல் பரிமாற்றங்களை உன்னிப்பாகக் கண்காணிப்பதன் மூலமும், சந்தைப்படுத்துபவர்களுக்கு மின்னஞ்சல் நுண்ணறிவு டாஷ்போர்டை வழங்குவதன் மூலமும் ஸ்மார்ட் மின்னஞ்சல் உதவியாளர் சந்தைப்படுத்துபவர்களுக்கு தங்கள் வாடிக்கையாளர்-களை நன்கு புரிந்துகொள்ள உதவுகிறது. மின்னஞ்சல் நுண்ணறிவு முக்கிய தேர்வு மற்றும் வகைப்படுத்தல், உரை சுருக்கம், உணர்வு பகுப்பாய்வு, உணர்ச்சி எதிர்வினை, நிறுவனம் பிரித்தெடுத்தல் மற்றும் வகைபிரித்தல் வகைப்பாடு ஆகியவை அடங்கும்.ஸ்மார்ட் மின்னஞ்சல் உதவியாளர் ஒரு சந்தைப்படுத்துபவரால் நிர்வகிக்கப்படும் முக்கிய வார்த்தைகளின் தொகுப்பிற்கான சூழல் வாக்கியத்தை பிரித்தெடுக்கும் திறனையும் வழங்குகிறது.

பயனர் மின்னஞ்சல்கள் மெயில்பார்சர் இன்பாக்ஸால் சேமிக்கப்பட்டு செயலாக்கப்படுகின்றன, இதில் பலபடி செயல்முறை மூலம், சேமித்தல், பாகுபடுத்துதல், தரவு பொருள்களை உருவாக்குதல் மற்றும் முன் கட்டமைக்கப்பட்ட வெப்ஹூக் அனுப்புதல் ஆகியவை அடங்கும். பாகுபடுத்தப்பட்ட மின்னஞ்சல் பின்னர் ஒரு HTTP பெறு முறை கோரிக்கையை மேற்கொள்வதன் மூலம் API கேட்வே வழியாக AWS லாம்ப்டாக்கு அனுப்பப்படுகிறது, மேலும் பதிலில் நுண்ணறிவுகளை உருவாக்குகிறது.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1. js – JavaScript.
2. AWS Lambda – Amazon Web Services.
3. API – Application Programming Interface.
4. REST – Representational State Transfer Protocol.
5. HTTP – HyperText Transfer Protocol.
6. CRM - Customer Relationship Management.
7. UCE –Unsolicited Cmmercial Email.
8. CSC – Customer Support Center.
9. HMM - Hidden Marvok Model.
10. DISC - Dominant,Inspiring,Supportive and Cautious.
11. SQL – Structured Query Language.
12. URL – Uniform Resource Locator.
13. AI – Artificial Intelligence.

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT:

Often marketers, receive a considerable amount of emails from users, and it is not very easy to track, understand, and take necessary actions to resolve a user's request without understanding the essence of an email. Email in its current form is fundamentally chaotic and difficult to extract from the confines of the inbox – which is what makes Smart Email Assistant**.**so useful .For example, if an email sent by a user is 200 words long, it would take a marketer minimum of 2 to 3 minutes or longer before taking any action, and thereby reducing a marketer's bandwidth to fewer email correspondences in a day.

1.2 PURPOSE:

The purpose of Smart Email Assistant is to help marketers to do more with less effort and focus on the things that matter most. Smart Email Assistant parse email data fields into sender's from details, sender's to details, email subject, and the actual email body. It allows the marketers to control the keywords they're searching for, and give back features such as keyword occurrence, keyword sentence as a response.

Smart Email Assistant also offers features, such as sentiment analysis, emotional reaction, entity extraction and taxonomy classification with less effort.

1.3 SCOPE OF PROJECT:

The project covers a wide scope. The emails from the users can be stored in MailParser and are categorized into original email, parsed email and dispatched webhooks as per the parsing rules. Parsed email is sent to A.W.S. Lambda through API Gateway and in response will give text summary to the marketer. Various list of modules included are:

- Mail Parser Mailbox Module
- Rule Parsing Module
- Parser Webhook API Module
- API Gateway and Lambda Function Module
- Lambda Function and DynamoDB Module
- Smart Email Assistant - Data Analytics Module (Core Functionality)
- Tableau - Data Insight Module

1.4 HARDWARE AND SOFTWARE SPECIFICATION

Operating system: A.W.S. (Amazon Web Services)

Languages: Node.js, SQL

Software:

- Amazon Web Services (A.W.S.) Lambda
- AWS API Gateway
- A.W.S. CloudWatch
- Mail Parser I/O
- Tableau
- MySQL
- Yonder Labs API

1.5 SOFTWARE DESCRIPTION

1.5.1 AWS Lambda

A.W.S. Lambda - Serverless Compute on Amazon Web Services:

A.W.S. Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying computing resources for you. You can use **A.W.S. Lambda** to extend other **A.W.S.** services with custom logic or create your own back-end services that operate at **A.W.S.** scale, performance, and security.

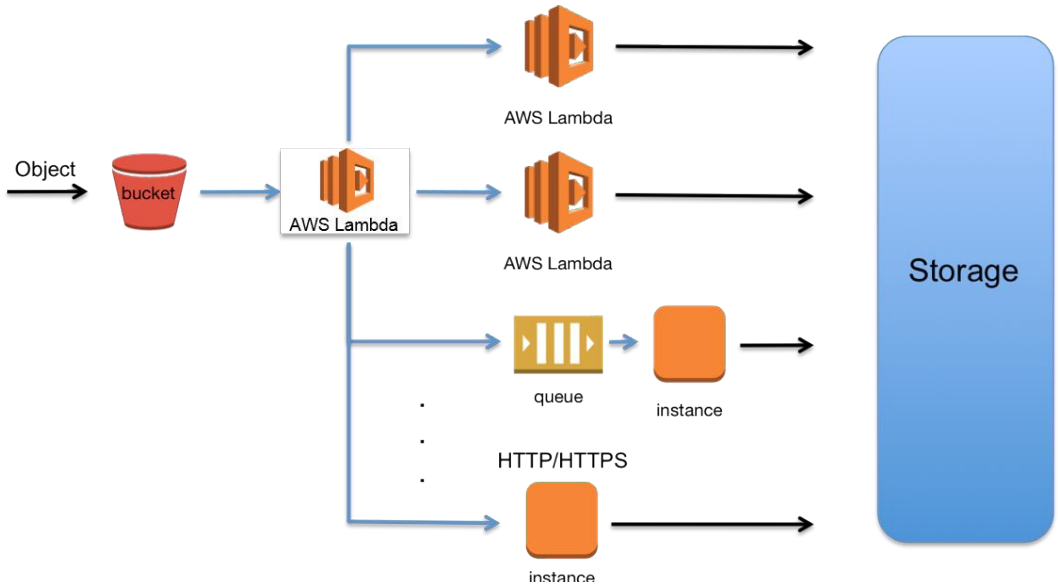


Fig.1 AWS Lambda

1.5.2 AWS API Gateway

Amazon API Gateway is an A.W.S. service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access A.W.S. or other web services, as well as data stored in the A.W.S. Cloud. As an API Gateway API developer, you can create APIs for use in your own client applications. Or you can make your APIs available to third-party app developers.

API Gateway creates RESTful APIs that:

- Are HTTP-based.
- Enable stateless client-server communication.
- Implement standard HTTP methods such as G.E.T., POST, P.U.T., PATCH, and DELETE.

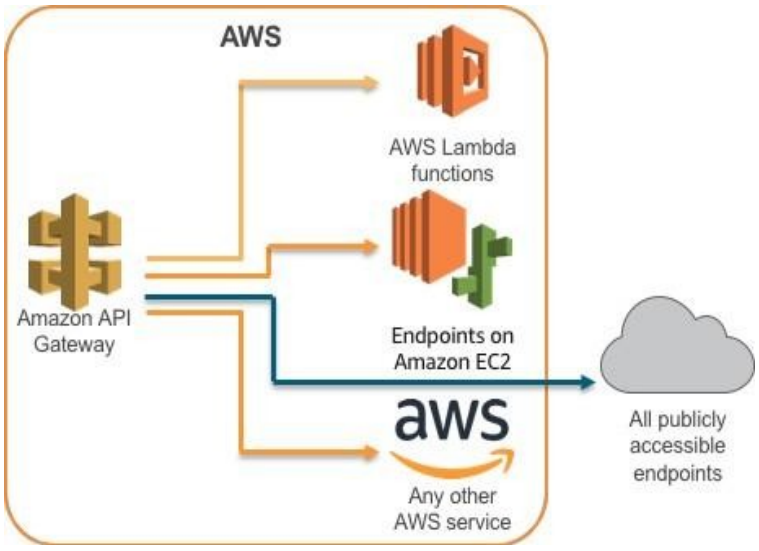


Fig.2 AWS API Gateway

1.5.3 AWS CloudWatch

Amazon **CloudWatch** is a monitoring and management service that provides data and actionable insights for **A.W.S.**, hybrid, and on-premises applications and infrastructure resources. With **CloudWatch**, you can collect and access all your performance and operational data in form of logs and metrics from a single platform.

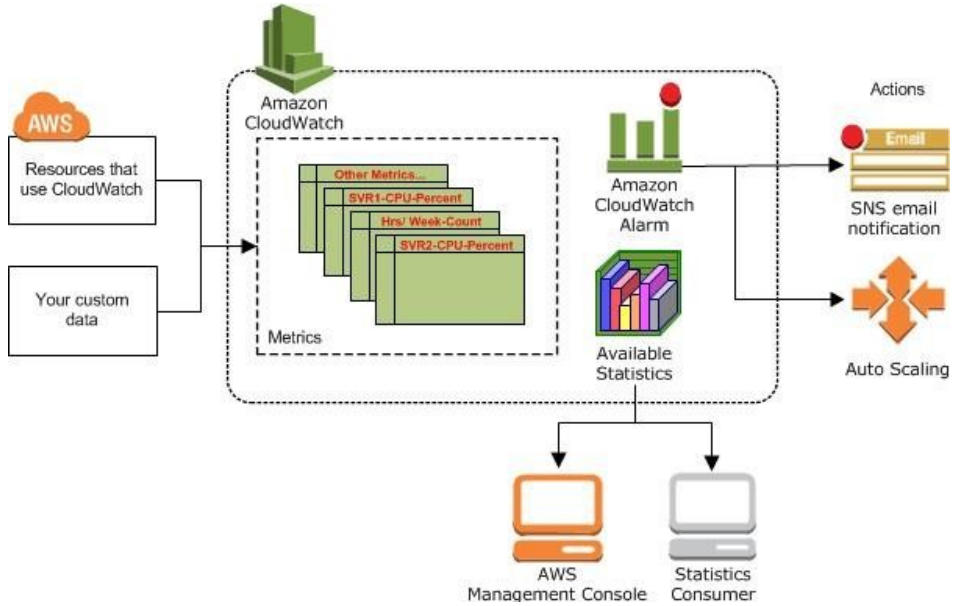


Fig.3 AWS CloudWatch

1.5.4 Mail Parser I/O

Mailparser.io is a data processing and workflow automation SaaS product for small and medium-sized online businesses. It provides powerful and reliable tools for extracting data from incoming emails and attachments and automatically transfers the parsed data to Google Sheets, Excel, your CRM, and hundreds of other third-party APIs. These features can save you plenty of time as you can avoid manual data entry and review while keeping the integrity of the data intact. Mailparser.io can process any kind of recurring email, for example, emails from contact forms, order receipts, etc.

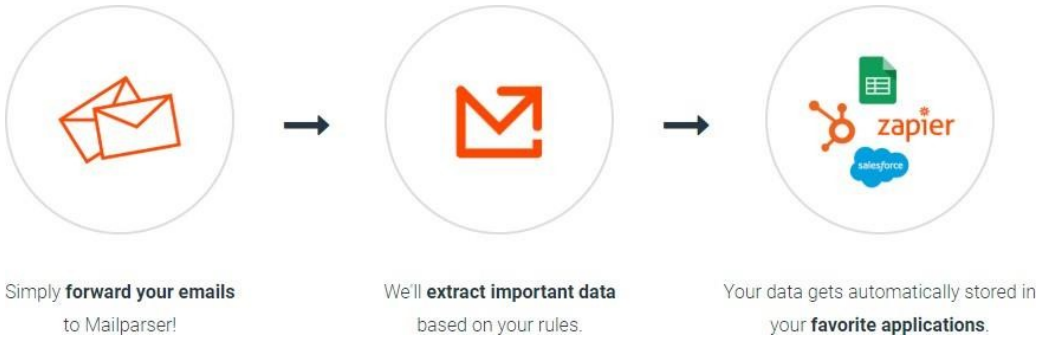


Fig.4 Mail Parser I/O

1.5.5 Tableau

Tableau Software is an American interactive data visualization software company. The company is currently focused on business intelligence. Tableau products query relational databases, online analytical processing cubes, cloud databases, and spreadsheets to generate graph-type data visualizations. The products can also extract, store, and retrieve data from an in-memory data engine.

Four types of DISC types:

- Dominant
- Inspiring,

- Supportive, and.
- Cautious

Fig.5 Tableau

1.5.6 MySQL

MySQL is a relational database management system based on SQL – Structured Query Language. The application is used for a wide range of purposes, including data warehousing, e-commerce, and logging applications. The most common use for **MySQL**, however, is for the purpose of a web database.

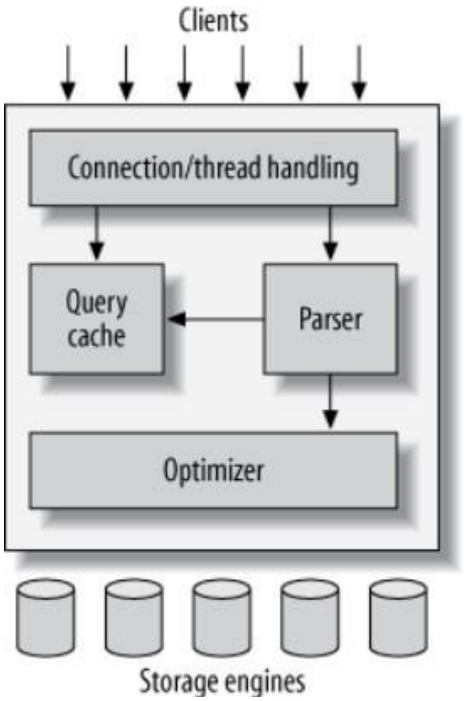


Fig.6 MySQL

1.5.7 Yonder Labs API

YonderAPI provides access to functionalities developed by Yonder. The following suite of text and image analysis API services can be composed together into a pipeline to solve a variety of problems and create smarter applications and services in different verticals.

The following set of API allows for extracting semantic information from Text Documents. If you need to create a Text Collection, refer to the below YonderAPI Text Collect:

Text Summarization

- **Produce a meaningful summary from your documents**

This API analyzes your text content and produces a summary containing meaningful facts, events, and relations.

Text Summarization - `POST fromText`

Attribute	Type	Description	Values
texts	string, required	the texts to be summarized	pass the texts between "" (e.g. "This is an example") or use url-encoded text

Fig.7 Text Summarization

Taxonomy Classification

- **Categorize your text into a hierarchical taxonomy**

This service analyzes chunks of text (from single sentences to full documents) and classifies them into customizable hierarchical taxonomies.

Taxonomy Classification - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") text

Fig.8 Taxonomy Classification

Entity Extraction

- **Identify people, places, and organizations in your text**

This API detects named entities and classify them as Person, Place, Organization or Misc. The service also links them to knowledge base repositories.

Entity Extraction - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") text

Fig.9 Entity Extraction

Keyword Extraction

- **Find meaningful keywords in your text document**

This API analyzes your text content identifying all meaningful keywords related to important facts, events and relations. Allowed input sources are: text content contained in an URL ("from URL") or just simple text.

Keyword Extraction - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") text

Fig.10 Keyword Extraction

Concept Tagging

- **Generate high-level semantic tags for your text document**

Concept tags are a limited and meaningful set of highly relevant named entities and important keywords related to facts, events, and relations found in your text. It is also possible to include custom whitelists and blacklists of concept tags upon request.

Concept Tagging - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") text

Fig.11 Concept Tagging

Sentiment Analysis

- **Identify positive or negative sentiments within text**

This API classifies the polarity of a given text at sentence and document level, identifying whether the expressed opinion is positive, negative, or neutral. Allowed input sources are: text content contained in an URL or just simple text.

Sentiment Analysis - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") text

Fig.12 Sentiment Analysis

Emotional Reaction

- **Predict emotions aroused from text such as angry, sad, etc.**

This API goes beyond polarity sentiment analysis and focus on emotional states likely to be aroused in the reader. In particular, the API identifies a maximum of two dominant emotions choosing among the popular Facebook reactions love, angry, Ahah, wow, sad.

Emotional Reaction - `POST fromText`

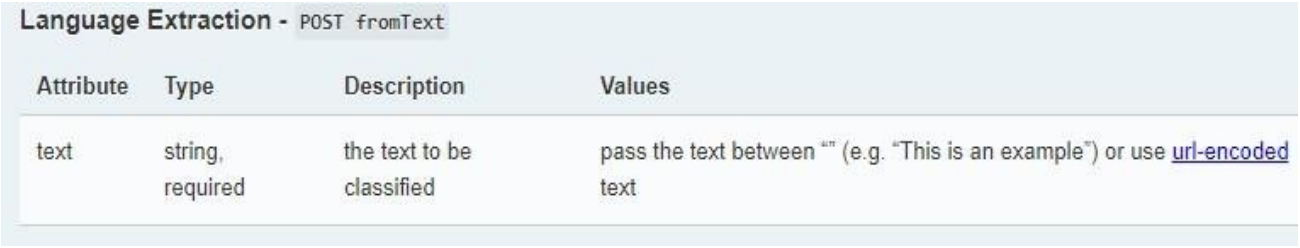
Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") or use url-encoded text

Fig.13 Emotional Reaction

Text Extraction

- **Extract text from a webpage removing all other undesired content**

This service extracts the main text from a webpage, discarding navigation links, advertisements, and other undesired content.



Language Extraction - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") or use url-encoded text

Fig.14 Text Extraction

Language Detection

- **Recognize the language of your text (over 50 languages)**

This service recognizes the language a document were written in, including English, Italian, Spanish, French, German, and other main languages. Allowed input sources are: text content contained in an URL ("from URL") or just simple text.

Language Extraction - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") or use url-encoded text

Fig.15 Language Detection

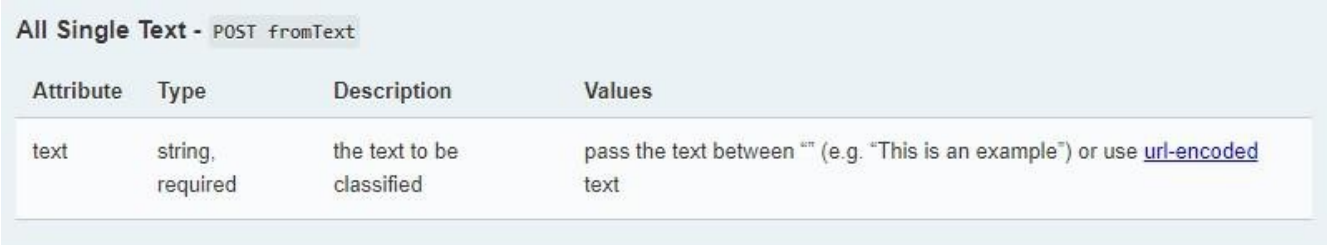
All Single Text

- **Analyze your document with all single API on text**

This service analyzes your text content by applying at once all single API on text

documents, that are:

- Text Extraction (fromURL only)
- Language Detection
- Taxonomy Classification
- Sentiment Analysis
- Emotional Reaction
- Entity Extraction
- Keyword Extraction
- Concept Tagging
- Text Summarization



All Single Text - `POST fromText`

Attribute	Type	Description	Values
text	string, required	the text to be classified	pass the text between "" (e.g. "This is an example") or use url-encoded text

Fig.16 All Single Text

1.6 DESCRIPTION OF LANGUAGE

1.6.1 FEATURES OF Node.js

- **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single-Threaded but Highly Scalable** − Node.js uses a single-threaded model with event looping. The event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers, which create limited threads to handle requests. Node.js uses a single-threaded program, and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.
- **License** − Node.js is released under the M.I.T. license

CHAPTER 2

LITERATURE SURVEY

2.1 Parsing of nested internet electronic mail documents

A system and method for automatically processing and responding to large volumes of customer complaints regarding Unsolicited Commercial Email (UCE) and other service disruptions. The complaints include one or more electronic mail (email) documents, each email document including a header and body portion. The process parses the header and body portions from each email document, normalizes the header and body portion by stripping unwanted characters, and extracts specific information relating to the email document from the source of the complaint. The extracted information can be input to a database that can be accessed by the Customer Support Center (CSC) of the Internet Service Provider (ISP).

2.2 Automated parsing of e-mail messages

An automated parser for e-mail messages identifies component parts such as header, body, signature, and disclaimer. The parser uses a hidden Markov model (HMM) in which the lines making up an e mail are treated as a sequence of observations of a system that evolves according to a Markov chain having states corresponding to the component parts. The HMM is trained using a manually-annotated set of e-mail messages, then applied to parse other e-mail messages. HMM-based parsing can be further refined or expanded using heuristic post-processing techniques that exploit redundancy of some component parts (e.g., signatures, disclaimers) across a corpus of e-mail messages.

2.3 Method and system for parsing e-mail

A method and system for parsing e-mail, and said system comprises a database and a

server. The server is used to receive a e-mail, then parse and extract the content of the received e-mail, and lastly store the data extracted from the e-mail into the database. The server further includes a parsing unit, a notifying unit and a setting unit. The parsing unit is used to parse the e-mail, and extract the data according to preset key words. The notifying unit is used to inform the original sender, who sent the e-mail, and the notifying unit will generate a notification when the parsing unit extracts the specified data. The setting unit is used to modify the key word setting used in the parsing unit.

2.4 Method and system for customizing e-mail transmissions based on content detection

A method and system for customizing e-mail transmissions based on content detection determines when an e-mail user likely intends to customize an e-mail message and prompts the user to customize the e-mail message if they have not done so. A parser parses the e-mail message (including the subject line) for clues that indicate that the sender likely intends to customize the message in a particular manner, then the e-mail program prompts the user to perform the customization. The prompt may be generated in response to the e-mail program detecting that customization settings have not been entered or may be generated unconditionally upon detecting a clue. The parser may also decompose sentences to provide matching of common phrases or meanings with phrases or meanings that indicate that the sender likely intends to perform a particular customization. The e-mail program may further “jump” to the appropriate input area for the customization in response to detecting an associated clue or the e-mail program may skip on or both of jumping and prompting in certain cases

CHAPTER 3

PROJECT DESCRIPTION

3.1 INTRODUCTION

In recent years, artificial intelligence (A.I.) has become an emerging trend in different fields: science, business, medicine, automotive, and education. A.I. has also reached marketing.

Email in its current form is fundamentally chaotic and difficult to extract from the confines of the inbox – which is what makes Smart Email Assistant (SEA) so useful.

With the enormous amount of incoming emails, marketers and sales team struggle to prioritize emails, thereby resulting in delivering a delayed response to some of the crucial emails which result in providing a poor customer experience. Smart Email Assistant provides a multi-step solution to give a better customer experience for a customer's email communications.

User emails are stored and processed by MailParser inboxes via a multi-step process which includes, saving the original email, parsing email data based on a set of user-defined parsing rules, creating necessary data objects, and invoking a pre-configured webhook dispatch. Parsed email is then sent to AWS Lambda through API Gateway by making an HTTP GET method request, and in response, the API Gateway provides an HTTP Response with the status code and the response body. When invoking the Lambda function, API gateway passes the formatted request object in JSON format. AWS Lambda processes the request, which contains email data and executes various steps to generate email insights for sales and marketing.

3.2 BLOCK DIAGRAM

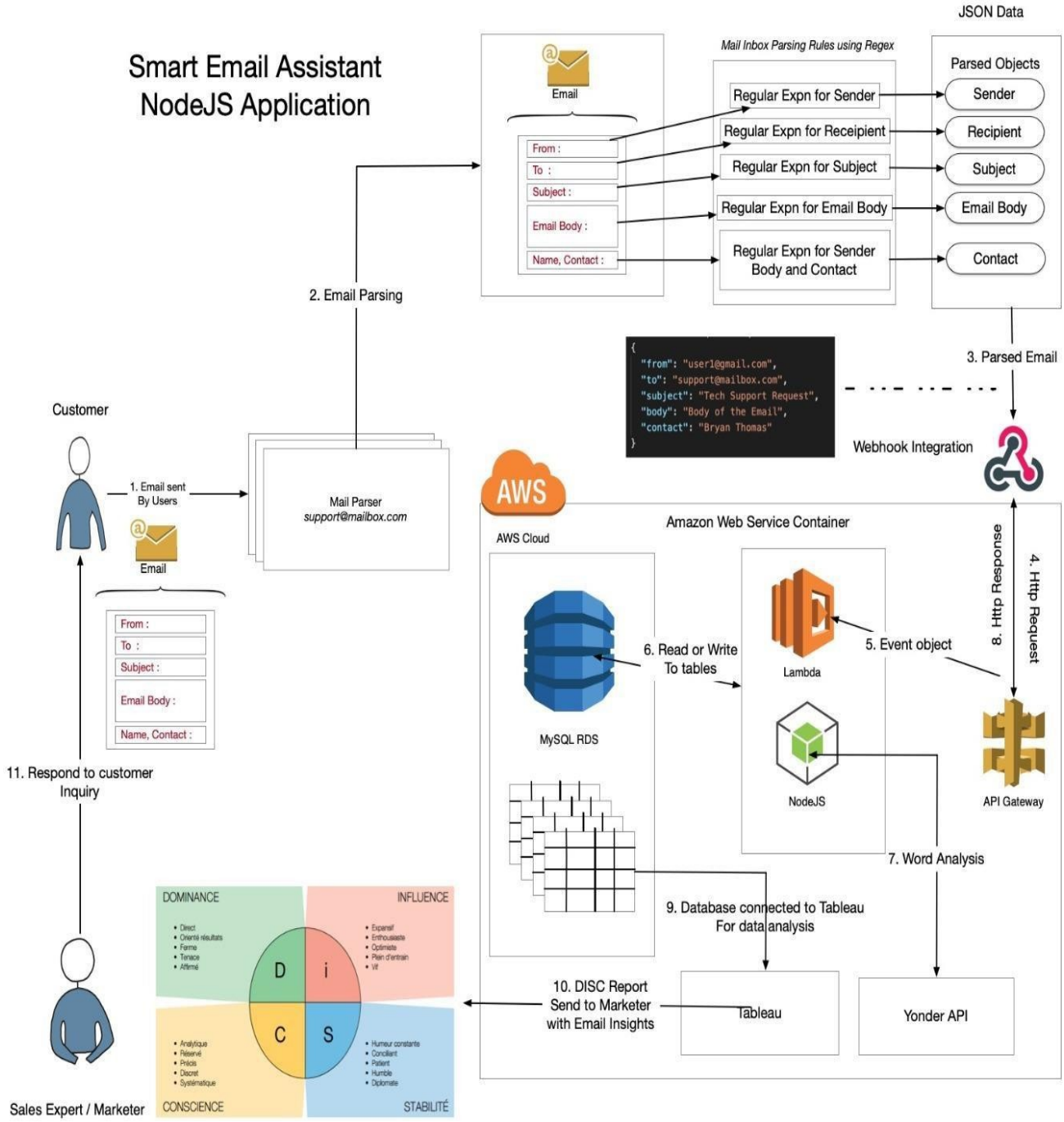


Fig.17 Block Diagram

3.3 SYSTEM STUDY AND ANALYSIS

3.3.1 PROBLEM DEFINITION

Often marketers, receive a considerable amount of emails from users, and it is not very easy to track, understand, and take necessary actions to resolve a user's request without understanding the essence of an email. Email in its current form is fundamentally chaotic and challenging to extract from the confines of the inbox – which is what makes Smart Email Assistant so useful . For example, if an email sent by a user is 200 words long, it would take a marketer minimum of 2 to 3 minutes or longer before taking any action, and thereby reducing a marketer's bandwidth to fewer email correspondences in a day.

3.3.2 PRODUCT INFORMATION

3.3.2.1 EXISTING SYSTEM:

- **Collecting the emails from the customers, flagging essential emails, and responding manually:**

In the sales and marketing sector, marketers usually receive all the emails from the customers and read (from top to bottom)manually to know the essence of it and then flag it if considered to be necessary.Then respond to them manually, which makes crucial and time-consuming and lower customers experience overall.

- **KnowMail:**

KnowMail learns your habits and finds the most important emails in your inbox based on those patterns. You can use it on your computer or a mobile device.

What Sets It Apart? The AI-powered interface goes beyond telling you how many urgent emails you need to read. It also gives you an estimated timeframe for doing so, making it easier to plan your day and stay productive

3.3.2.2 PROPOSED SYSTEM:

Steps to be followed to create a proposed system:

1. **Create an incoming email address** - you would simply create an incoming email address on Parserr for your company to send these emails to.
2. **Create a number of rules** - rules define how the incoming emails will be parsed and the relevant data extracted
3. **Use the data**- this is where it gets powerful! Using MailParser, you can link Parserr to pretty much any application. That means you could send your email data to any 3rd party app.
4. The targeted URL reaches the AWS Lambda via API gateway.
5. Parsed email is then sent to AWS Lambda through API Gateway by making a HTTP GET method request, and in response the API Gateway provides a HTTP Response with the status code and the response body.
6. When invoking the Lambda function, API gateway passes the formatted request object in JSON format. AWS Lambda processes the request which contains email data and executes various steps to generate email insights for sales and marketing.

3.3.3 PRODUCT FEATURES

- Complete automation is possible in this sector, which is against the main disadvantage, namely time-consuming.
- Can maintain customer details who have been reporting via mail.
- Sentence of the keyword can be retrieved to know the exact meaning or reaction of the keyword.
- It provides solutions to understand customers better.
- Also provides features such as :
    - keyword selection and categorization
    - text summarization

- sentiment analysis
- emotional reaction
- entity extraction and
- taxonomy classification

3.3.4 REQUIREMENT ANALYSIS

3.3.4.1 FUNCTIONAL REQUIREMENTS:

A customer should be able to login to their email accounts and mention their subject, the content of the mail, body of the mail. A marketer can log in to their account and forward this email to the MailParser email id.

3.3.4.2 NON FUNCTIONAL REQUIREMENTS:

Usability

This section includes all of those requirements that affect usability.

- We get the response within seconds.
- This software has a simple, user-friendly interface so marketers can save time.

Reliability

- The system is more reliable. The code built by using Node.js is more reliable.

Supportability

- The system is designed to be the cross-platform supportable. The system is supported on a wide range of hardware, and any software platform as this application is being developed using AWS Lambda.

Implementation

- The system is implemented in the AWS Cloud environment.

3.3.5 MODULES DESCRIPTION

3.3.5.1 Mail Parser Mailbox Module

- Sent a sample email from the client to the server.
- The original email in its current form from the users is forwarded directly to the mail

parser email id.

- User emails are stored initially and processed by MailParser inboxes.

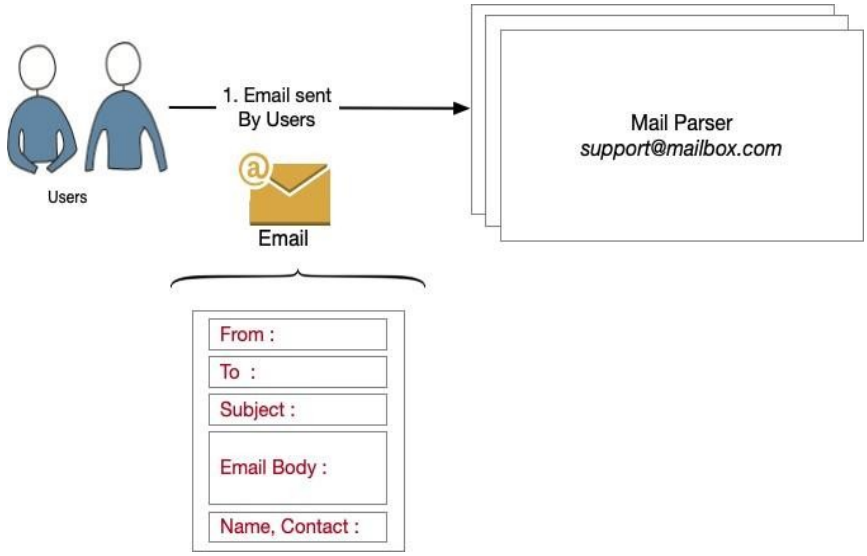


Fig.18 Mail Parser Mailbox Module

3.3.5.2 Rule Parsing Module

- The original user mails are stored in the mail parser mailbox.
- Parsing the user's mail.
- The parsed user email is processed into objects separately by using a set of parsing rules.

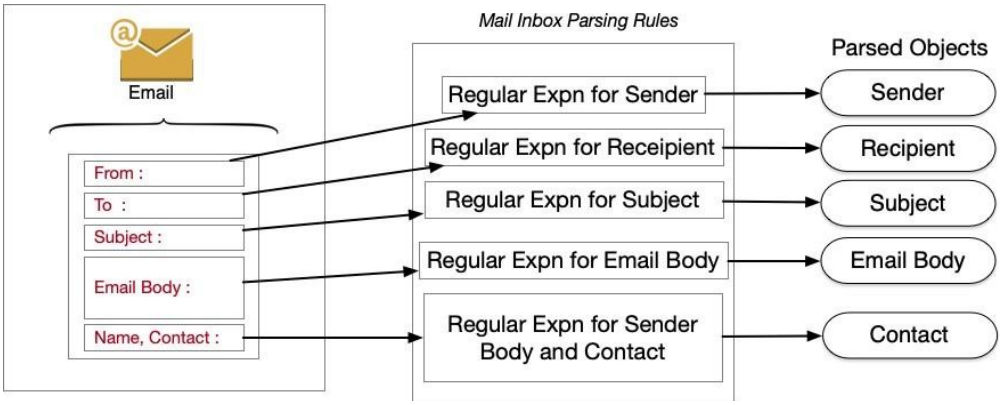


Fig.19 Rule Parsing Module

3.3.5.3 Parser Webhook API Module

- The parsed objects are converted into JSON format.
- The JSON format parsed objects are sent to the targeted URL by invoking a pre-configured webhook integrations.
- The targeted URL reaches to the AWS Lambda via API gateway.

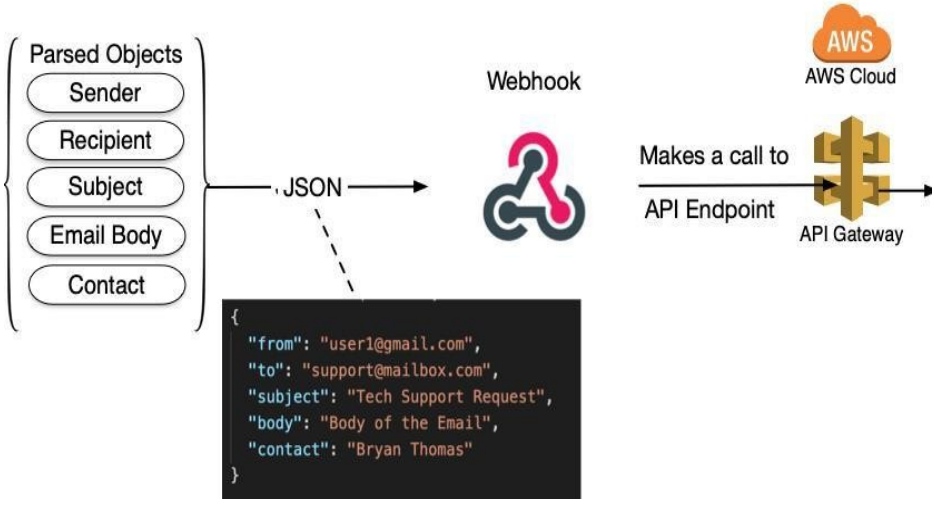


Fig.20 Parser Webhook API Module

3.3.5.4 API Gateway and Lambda Function Module

- The targeted URL reaches the AWS Lambda via API gateway.

- Parsed email is then sent to AWS Lambda through API Gateway by making an HTTP GET method request, and in response the API Gateway provides an HTTP Response with the status code and the response body.
- When invoking the Lambda function, API gateway passes the formatted request object in JSON format. AWS Lambda processes the request which contains email data and executes various steps to generate email insights for sales and marketing.

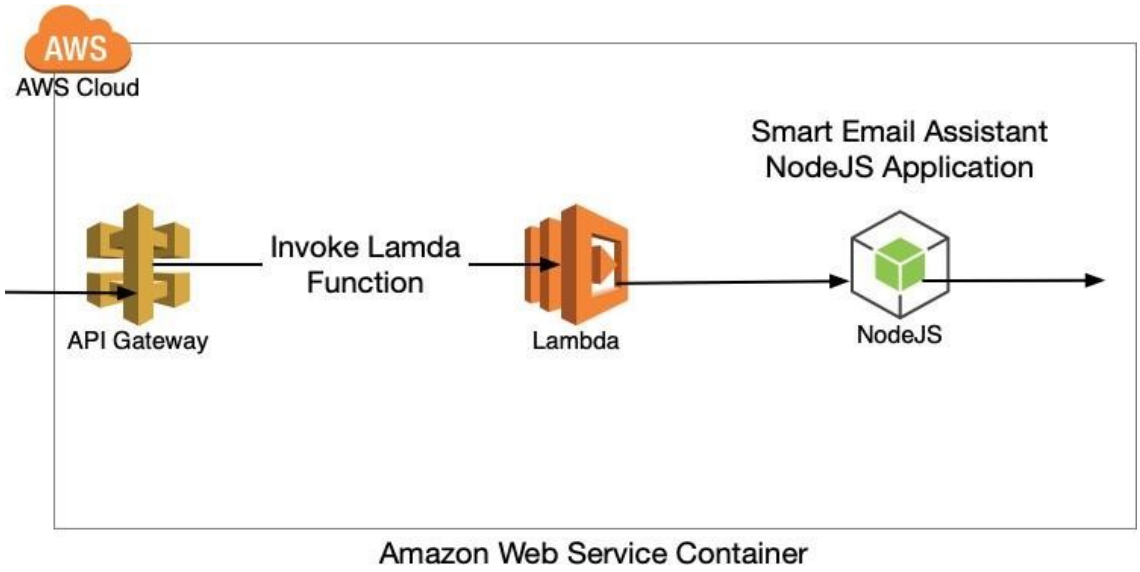


Fig.21 API Gateway and Lambda Function Module

3.5.5 Lambda Function and DynamoDB Module

- API Gateway forwards the request to lambda function for request processing.
- Lambda function receives the request, invokes smart email assistant application.
- Smart Email Assistant processes the request and its associated content (JSON).
- SEA populates necessary database tables after executing the algorithm.
- SEA then creates a response body and sends it API Gateway with response body and a response code.

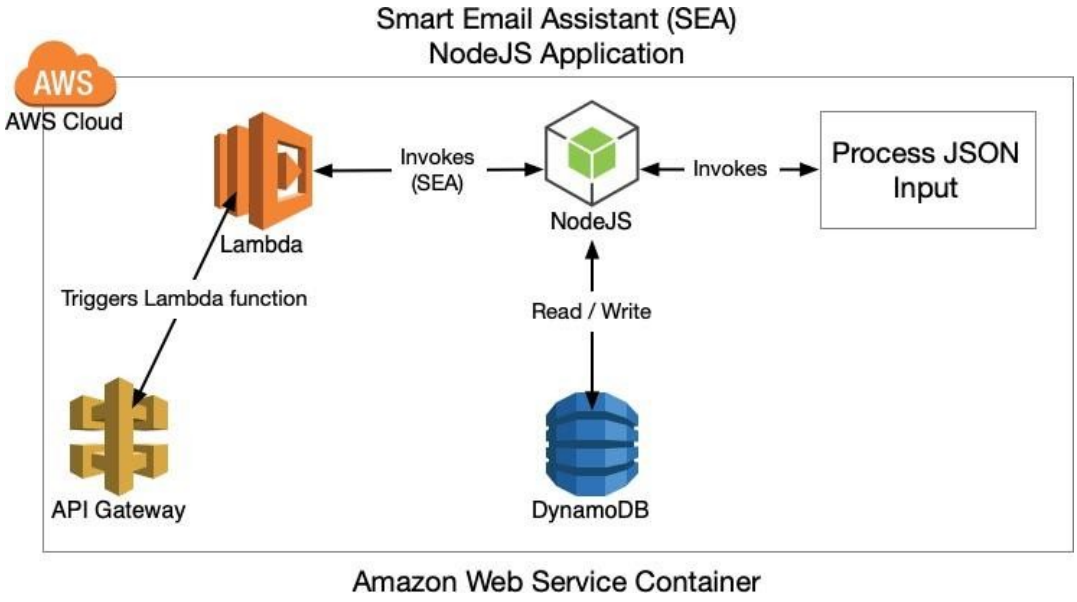


Fig.22 Lambda Function and DynamoDB Module

3.3.5.6 Smart Email Assistant - Data Analysis Module (Core Functionality)

- Lambda function (SEA Core Logic) receives the HTTP request from mail parser and executes a set of checks.
- Performs a data cleaning operation for the incoming email.
- Performs a data preparation for email body parsing
- Stores email data to EMAIL table
- Performs a keyword count for the pre-defined keywords stored in the KEYWORD table and updates the COUNT table.
- Updates the keyword sentence table with keywords and its context sentences.
- Makes a yonder api call with the email body, and updates the following tables
  - Email_emotion (Emotional Reaction)
  - Emai_keywords (Keyword Extraction)
  - Email_entities (Entity Extraction)

- Email_sentiment (Sentiment Analysis)
- Email_taxonomy (Taxonomy Classification)
- Email_summary (Text Summarization)
- Email_tags (Concept Tagging)

- After successfully processing the HTTP Request from API Gateway, a response is created with a status code and response body.
- If the algorithms fail or if any of the essentials parameters are not passed along with the HTTP request, an error code and a message is retuned to the API gateway as a response.

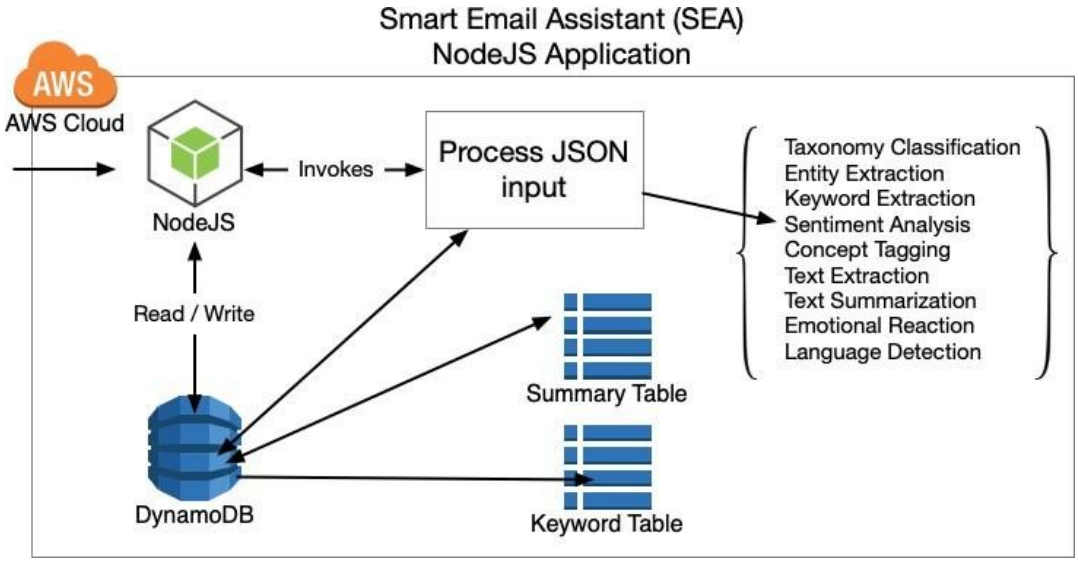


Fig.23 Smart Email Assistant - Data Analysis Module (Core Functionality)

3.3.5.7 Tableau - Data Insight Module

- Tableau dashboard reads data from the database and creates a graph report for marketers.
- Insights obtained from the tableau report is then used by a marketer to prioritize the tasks, and then responds to a customer's inquiry.

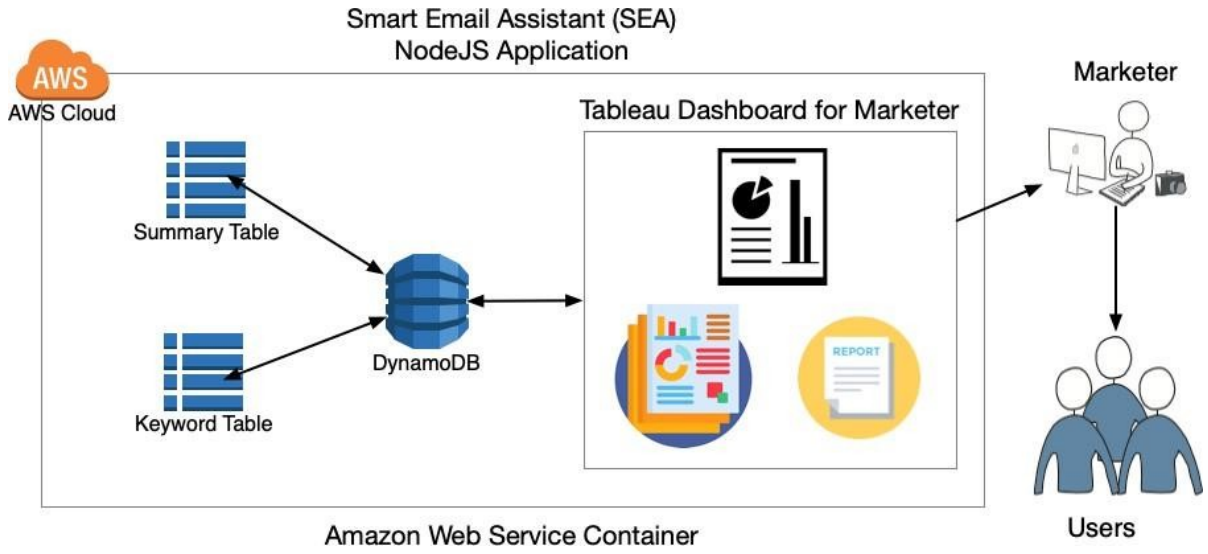


Fig.24 Tableau - Data Insight Module

APPENDIX

```
'use strict';//not to use undeclared variables

var mysql = require('mysql');//pkg
var request = require("request");//pkg

var eventRequest = null;//fut

var emailSentences = null;//fut

// server:Receives the request and handles response for the API Call
//event-request,context-invocation information,callback-response/err/promise
exports.handler = function(event, context, callback) {

    //var event = JSON.parse(JSON.stringify(event));
    eventRequest = event;
    callAPI();//fnc


    // Generate a greeting
    let greeting = "Good Morning";


    // convert the email body into lower case for keyword comparison

    if (event.body_of_email != null) {

        // Email Sentence Array
        //data cleaning-noise removal
        var sentenceEmailBody = event.body_of_email;
        //regular exp for replacing "  to ' to avoid confusion to the compiler.
        sentenceEmailBody = sentenceEmailBody.replace(/\"/g, "'");

        //console.info(sentenceEmailBody);

        emailSentences = sentenceEmailBody.split(/(?<=(?:\/|\.|\?|!))/);//body splitted into each sentence array


        event.body_of_email = event.body_of_email.toLowerCase();//data preparation for keyword table comaprsion
        event.body_of_email = event.body_of_email.replace("\"", "'");//replacing \ to null

        // Store the email into an array of sentences. Used for adding 2 sentences for each keyword occurence
        // Regex to capture sentences ending with .,?,!

        var sentenceSplitRegex = new RegExp(`[.?!]`, 'g');//g- global search
        // removes extra dot's in an email
        event.body_of_email = event.body_of_email.replace(/[.]{2,}/g, ".");//replace ... to .

        // 1. Generate update statement for email table
        var sqlQuery2 = updateEmail(event);//fut
        var connection = invokeConnection();//fut

        // 2. Invoke the sql connection and update email table
        connection.query(sqlQuery2, function(error, results, fields) {
            if (error) {
                connection.destroy();
                throw error;
            }
            else {
                callback(error, results);
                connection.end();
            }
        });

        // 3a. Query the keyword table and invoke a parse email function
        // 3a. Generates the keyword count
        // 3b. Generate Multi SQL query for all keywords
        queryKeyword(event);


    }
    else {
        greeting = "Oops, Please try again :-)   ";
        console.info("CONSOLE ERROR: Empty Email Body");
        console.info("Email Parser ID = " + event.id);
        console.info("Email MailBox ID = " + event.mail_box_id);
    }

    // Return a greeting to the caller
    callback(null, {
        "greeting": greeting
    });
};

/*
 * Function to generate the Email SQL Update statement based off the request from mailbox parser
 * Returns - SQL query
 */
function updateEmail(event) {

    var email_parser_id = event.id == null ? 911 : event.id;
    var email_mailbox_id = event.mail_box_id == null ? 911 : event.mail_box_id;
    var email_date = event.date == null ? 911 : event.date;
    var email_subject = event.mail_subject == null ? 911 : event.mail_subject.replace(/\"/g, "'");
    var email_from = event.forwarded_from_address == null ? 911 : event.forwarded_from_address;
    var email_to = event.forwarded_to_address == null ? 911 : event.forwarded_to_address;
    var email_body = event.body_of_email == null ? 911 : event.body_of_email.replace(/\"/g, "'");
    var queryEmailSummary = "insert into `email` VALUES (\"" + email_parser_id + "\", \"" + email_mailbox_id + "\"

    console.info("Email Parser ID = " + email_parser_id);
    console.info("Email MailBox ID = " + email_mailbox_id);

    return queryEmailSummary;
}

/*
 * a. Function to query the keyword table and parse the body of the email to count word occurences
 * b. Genrates a SQL statement for each keyword for which an occurence value is greater than 0
 * c. Invokes the updateCountTable function to update the count for each keyword occurence
 */
async function queryKeyword(event) {
    var insertQuery = "";

    var connection1 = invokeConnection();

    connection1.query("select * from keyword", function(error, results) {
        if (error) {
            connection1.destroy();
            throw error;
        }
        else {

            //getting current date and time
            var today = new Date();
            var date = today.getFullYear() + '-' + (today.getMonth() + 1) + '-' + today.getDate();
            var time = today.getHours() + ":" + today.getMinutes() + ":" + today.getSeconds();
            var dateTime = date + ' ' + time;


            // Checking each keyword against the body of the email
            var keyword;
            var i = 0;
            //console.info("Inside here");
            for (keyword of results) {
                i++;
                var re = new RegExp(`\\b${keyword.keyword_text}\\b`, 'gi');//re to check globally from key
                var count = event.body_of_email.match(re);//matches re with email body
                if (count != null) {
                    var sentenceValue = "";
                    var k = 1;

                    for (var j = 0; j < emailSentences.length; j++) {
                        if (emailSentences[j].match(re) != null) {


                            //console.info(keyword.keyword_text);

                            var sentence2Split = emailSentences[j].split(re);
                            var sentence1 = emailSentences[j - 1] = undefined ? "----" : emailSentences[j - 1];
                            var sentence2 = sentence2Split[0];
                            var sentence3 = sentence2Split[1];
                            var sentence4 = emailSentences[j + 1] == undefined ? "----" : emailSentences[j + 1];


                            insertQuery += "insert into `count` VALUES (\"" + event.id + "\", \"" + event.date + "\"
                            k++;

                        }
                    }
                }
                if (i == results.length) {

                    var insertEmptyQuery = "insert into `count` VALUES (\"" + event.id + "\", \"" + event.date + "\"
                    insertQuery = insertQuery == "" ? insertEmptyQuery : insertQuery;
                    updateCountTable(insertQuery);
                }
            }


            connection1.end();
        }
    });
}

/*...

function updateCountTable(insertQuery) {
    // Creating a Multi Statement AWS RDS Connection object
    var connection2 = invokeConnection();
    connection2.query(insertQuery, function(error, results, fields) {
        if (error) {
            connection2.destroy();
            throw error;
        }
        else {
            connection2.end();
        }
    });
}

/*...

function invokeConnection() {
```

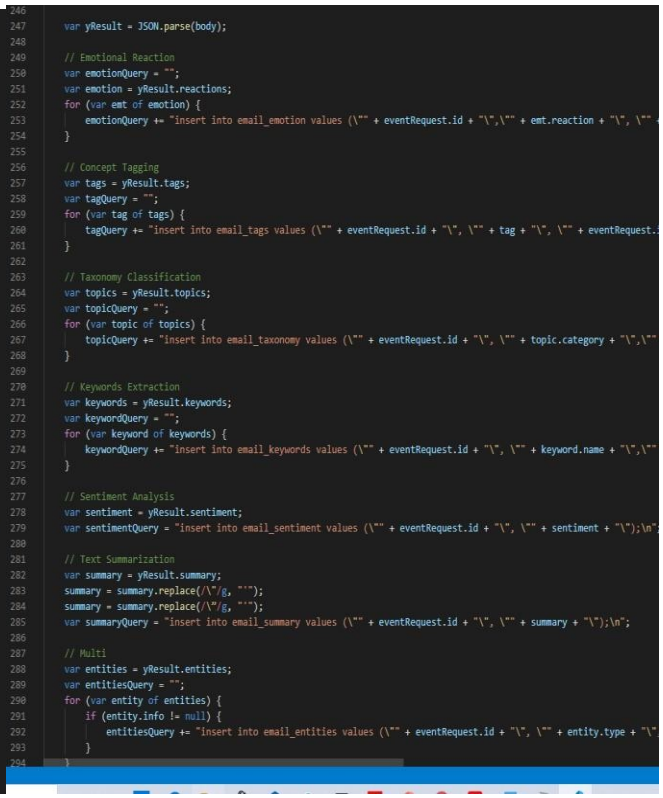
```
function invokeConnection() {
    // Creating a Multi Statement AWS RDS Connection object
    var conn = mysql.createConnection({
        host: "clearpathgps.cm8pentb5trt.us-east-1.rds.amazonaws.com",
        user: "clearpathgps",
        password: "clearpathgps",
        database: "clearpathgps",
        multipleStatements: true
    });
    return conn;
}


async function callAPI() {

    if (eventRequest.body_of_email != null) {
        eventRequest.body_of_email = eventRequest.body_of_email.replace("\"", "'");
        var test = eventRequest.body_of_email;

        var options = {
            method: 'POST',
            url: 'https://api.yonderlabs.com/1.0/text/allsingletext/fromText',
            qs: {
                nwords: '100',
                taxonomy: 'news-en',
                levels_taxonomy: '1',
                limit_taxonomy: '3',
                limit_entity: '10',
                limit_keyword: '12',
                limit_concept: '10',
                access_token: 'c1670c6a69e03d055bde5975c98dd7ef3188fe0cfd35dac62d30d488100904a5'
            },
            formData: { text: test }
        };

        request(options, function(error, response, body) {
            if (error) throw new Error(error);
            yonderQuery(body);
        });
    }else{
        console.info("CONSOLE ERROR: Yonder Empty Email Body");
        console.info("Email Parser ID = " + eventRequest.id);
        console.info("Email MailBox ID = " + eventRequest.mail_box_id);
    }
}
```



```
    var yResult = JSON.parse(body);

    // Emotional Reaction
    var emotionQuery = "";
    var emotion = yResult.reactions;
    for (var emt of emotion) {
        emotionQuery += "insert into email_emotion values (\"" + eventRequest.id + "\",\"" + emt.reaction + "\", \"" +
    }

    // Concept Tagging
    var tags = yResult.tags;
    var tagQuery = "";
    for (var tag of tags) {
        tagQuery += "insert into email_tags values (\"" + eventRequest.id + "\", \"" + tag + "\", \"" + eventRequest.
    }

    // Taxonomy Classification
    var topics = yResult.topics;
    var topicQuery = "";
    for (var topic of topics) {
        topicQuery += "insert into email_taxonomy values (\"" + eventRequest.id + "\", \"" + topic.category + "\",\""
    }

    // Keywords Extraction
    var keywords = yResult.keywords;
    var keywordQuery = "";
    for (var keyword of keywords) {
        keywordQuery += "insert into email_keywords values (\"" + eventRequest.id + "\", \"" + keyword.name + "\",\""
    }

    // Sentiment Analysis
    var sentiment = yResult.sentiment;
    var sentimentQuery = "insert into email_sentiment values (\"" + eventRequest.id + "\", \"" + sentiment + "\");\n"

    // Text Summarization
    var summary = yResult.summary;
    summary = summary.replace(/\"/g, "'");
    summary = summary.replace(/\"/g, "'");
    var summaryQuery = "insert into email_summary values (\"" + eventRequest.id + "\", \"" + summary + "\");\n";

    // Multi
    var entities = yResult.entities;
    var entitiesQuery = "";
    for (var entity of entities) {
        if (entity.info != null) {
            entitiesQuery += "insert into email_entities values (\"" + eventRequest.id + "\", \"" + entity.type + "\"
        }
    }
```

Create table commands:

```
CREATE TABLE `count` (
  `count_parser_id` varchar(100) NOT NULL,
  `count_parser_date` varchar(100) DEFAULT NULL,
  `count_related_keyword ID` varchar(100) NOT NULL,
  `count_keyword_designator` varchar(100) DEFAULT NULL,
  `count_keyword_text` varchar(100) DEFAULT NULL,
  `count_keyword_wordgroup` varchar(100) DEFAULT NULL,
  `count_created` varchar(100) DEFAULT NULL,
  `count_modified` varchar(100) DEFAULT NULL,
  `maild_integer` varchar(100) DEFAULT NULL,
  `occurence` int(11) NOT NULL,
  `sentence1` text,
  `sentence2` text,
  `sentence3` text,
  `sentence4` text,
  PRIMARY KEY (`count_parser_id`,`count_related_keyword ID`,`occurence`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `directory` (
  `directoryid` int(11) NOT NULL,
  `directory_email` varchar(45) DEFAULT NULL,
  `directory_first_name` varchar(45) DEFAULT NULL,
  `directory_last_name` varchar(45) DEFAULT NULL,
  `directory_full_name` varchar(45) DEFAULT NULL,
  `directory_company` varchar(45) DEFAULT NULL,
  `directory_title` varchar(45) DEFAULT NULL,
  `directory_address` varchar(45) DEFAULT NULL,
  `directory_city` varchar(45) DEFAULT NULL,
  `directory_zip` varchar(45) DEFAULT NULL,
  `directory_function` varchar(45) DEFAULT NULL,
  `directory_phone` varchar(45) DEFAULT NULL,
  `directory_address_city` varchar(45) DEFAULT NULL,
  `directory_address_country` varchar(45) DEFAULT NULL,
  `directory_address_postal_code` varchar(45) DEFAULT NULL,
  `directory_address_state_region` varchar(45) DEFAULT NULL,
  `directory_address_street1` varchar(45) DEFAULT NULL,
  `directory_address_street2` varchar(45) DEFAULT NULL,
  `related_parser_id` varchar(45) DEFAULT NULL,
  `parser_id_from` varchar(45) DEFAULT NULL,
  `parser_id_to` varchar(45) DEFAULT NULL,
  `date_created` varchar(45) DEFAULT NULL,
  `date_modified` varchar(45) DEFAULT NULL,
  `last_modified_by` varchar(45) DEFAULT NULL,
  `record_owner` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`directoryid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `email` (
  `email_parser_id` varchar(100) NOT NULL,
  `email_mailbox_id` varchar(100) DEFAULT NULL,
  `email_date` varchar(45) DEFAULT NULL,
  `email_subject` varchar(100) DEFAULT NULL,
  `email_from` varchar(100) DEFAULT NULL,
  `email_to` varchar(100) DEFAULT NULL,
  `email_body` text,
  PRIMARY KEY (`email_parser_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `email_emotion` (
  `emotion_email_id` varchar(100) NOT NULL,
  `emotion_name` varchar(100) NOT NULL,
  `emotion_score` float DEFAULT NULL,
  `emotion_key` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emotion_email_id`,`emotion_name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `email_entities` (
  `ent_email_id` varchar(100) DEFAULT NULL,
  `ent_type` varchar(100) DEFAULT NULL,
  `ent_name` varchar(100) DEFAULT NULL,
  `ent_score` float DEFAULT NULL,
  `ent_key` varchar(100) NOT NULL,
  PRIMARY KEY (`ent_key`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `email_keywords` (
  `key_email_id` varchar(100) DEFAULT NULL,
  `key_name` varchar(100) DEFAULT NULL,
  `key_score` float DEFAULT NULL,
  `key_key` varchar(100) NOT NULL,
  PRIMARY KEY (`key_key`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `email_sentiment` (
  `sentiment_email_id` varchar(100) NOT NULL,
  `sentiment_name` varchar(100) NOT NULL,
  PRIMARY KEY (`sentiment_email_id`,`sentiment_name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `email_summary` (
  `sum_email_id` varchar(100) NOT NULL,
  `sum_body_summary` text,
  PRIMARY KEY (`sum_email_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
  `tax_key` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`tax_email_id`,`tax_category`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `guidance` (
  `guidance_word_group` varchar(100) DEFAULT NULL,
  `guidance_text` text,
  `Date Created` varchar(100) DEFAULT NULL,
  `Date Modified` varchar(100) DEFAULT NULL,
  `Record ID#` varchar(100) DEFAULT NULL,
  `Record Owner` varchar(100) DEFAULT NULL,
  `Last Modified By` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `keyword` (
  `keyword_id` int(11) NOT NULL,
  `keyword_word_group` varchar(100) DEFAULT NULL,
  `keyword_designator` varchar(100) NOT NULL,
  `keyword_text` varchar(100) DEFAULT NULL,
  `keyword_created` varchar(100) DEFAULT NULL,
  `keyword_modified` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`keyword_id`,`keyword_designator`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `keywordSentence` (
  `emailid` int(11) NOT NULL,
  `keyword` varchar(45) NOT NULL,
  `sentence1` varchar(200) DEFAULT NULL,
  `sentence2` varchar(200) DEFAULT NULL,
  `sentence3` varchar(200) DEFAULT NULL,
  `sentence4` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`emailid`,`keyword`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `mailbox` (
  `mailbox_id` int(11) NOT NULL,
  `mailbox.org` varchar(45) DEFAULT NULL,
  `Date Created` varchar(45) DEFAULT NULL,
  `Date Modified` varchar(45) DEFAULT NULL,
  `Mailbox Record ID#` varchar(45) DEFAULT NULL,
  `Record Owner` varchar(45) DEFAULT NULL,
  `Last Modified By` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`mailbox_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

CHAPTER 4

RESULT
MAILPARSSER

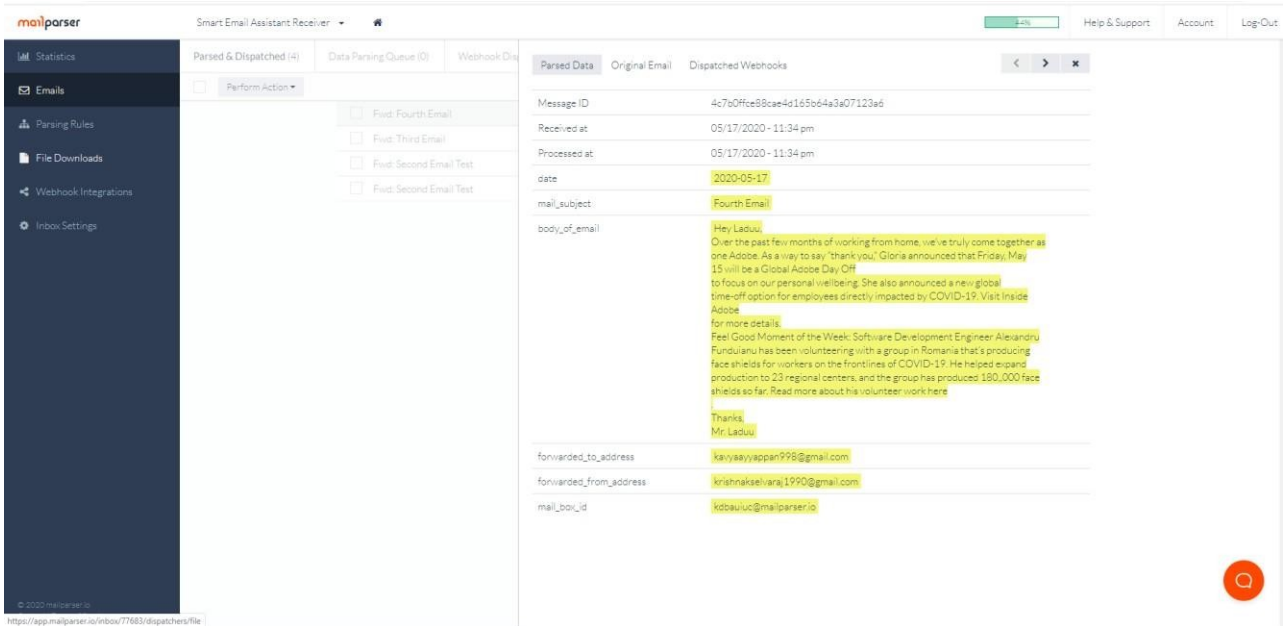


Fig.25 MAILPARSSER

AWS CLOUDWATCH LOG

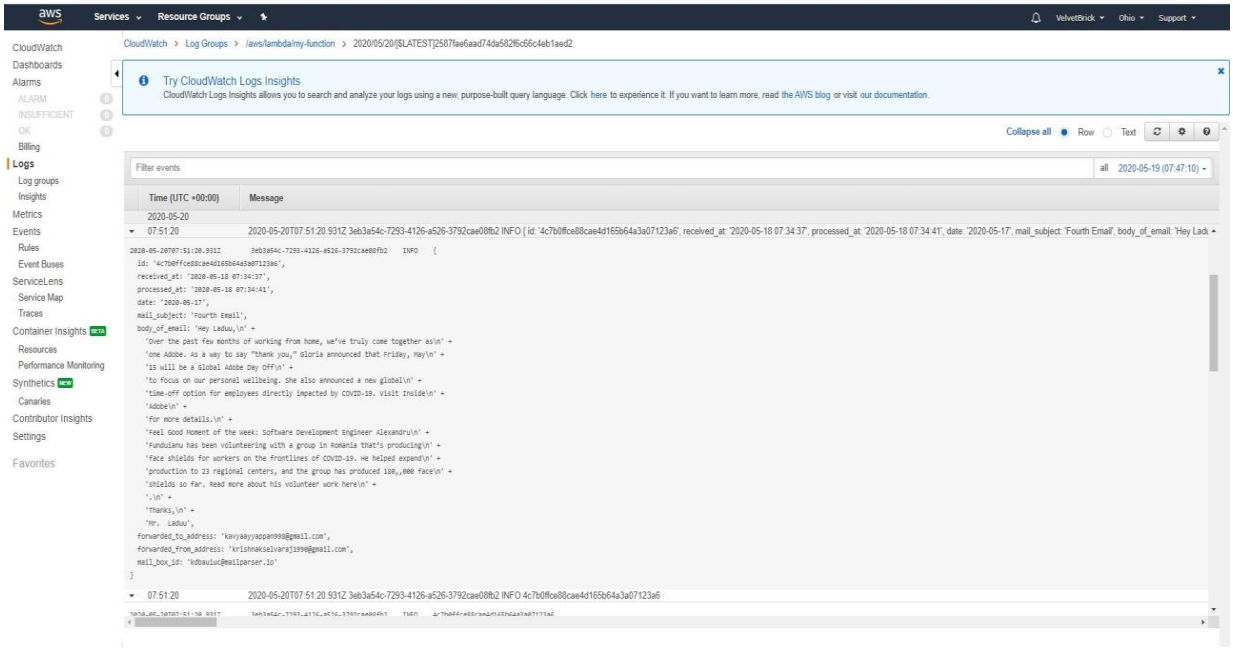


Fig.26 AWS CLOUDWATCH LOG

MYSQL WORKBENCH

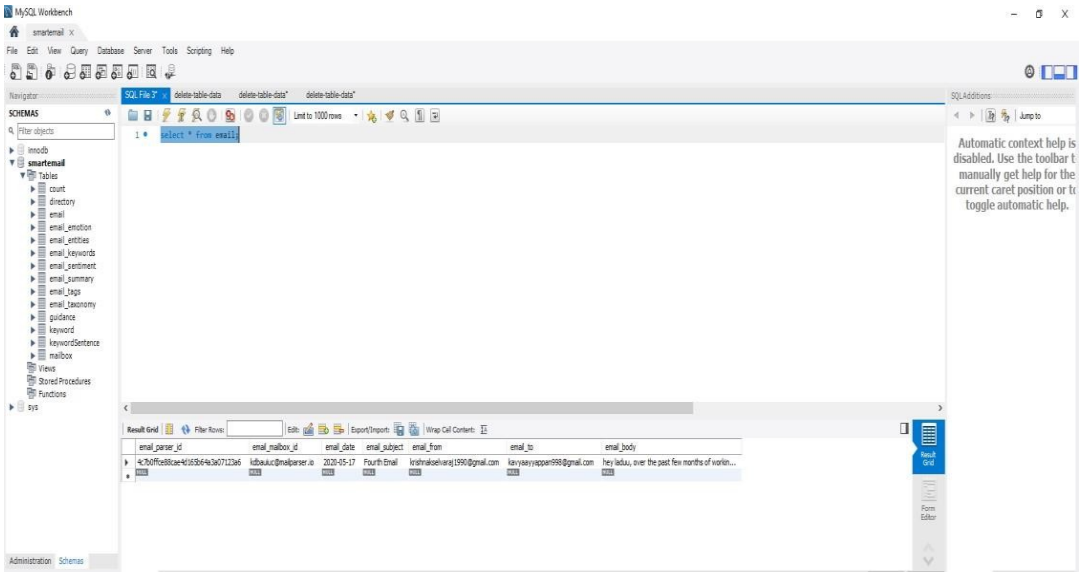


Fig.27 MYSQL WORKBENCH

TABLEAU

Fig.28 TABLEAU

CHAPTER 5

5.1 CONCLUSION

Smart Email Assistant provides a multi-step solution to give a better customer experience for a customer's email communications. User emails are stored and processed by MailParser inboxes via a multi-step process which includes, saving the original email, parsing email data based on a set of user-defined parsing rules, creating necessary data objects, and invoking a pre-configured webhook dispatch. Parsed email is then sent to AWS Lambda through API Gateway by making an HTTP GET method request, and in response, the API Gateway provides an HTTP Response with the status code and the response body.

FUTURE SCOPE

The Smart Email Assistant (SEA) currently performs a DISC analysis for a customer's email exchange with a sales or a support team. With the help of the DISC analysis report and data insights generated by the application, a sales executive can then deliver better correspondence to ensure customer satisfaction and improve overall sales revenue. In the future, Smart Email Assistant could be used for understanding customer behavior in different marketing channels like phone calls messages.

REFERENCES

1. http://yonderlabs.github.io/api-docs/#intro-to-yonderapi
2. file:///C:/Program%20Files/WindowsApps/5319275A.WhatsAppDesktop_2.2019.8.0_x64__cv1g1gvanyjgm/app/resources/app.asar/index.html#
3. https://patents.google.com/patent/US7103599B2/en
4. https://patents.google.com/patent/US8527436B2/en
5. https://patents.google.com/patent/US7174366B2/en
6. https://patents.google.com/patent/US7092993B2/en
7. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html
8. https://docs.aws.amazon.com/lambda/latest/dg/lambda-nodejs.html
9. https://app.mailparser.io/dispatcher/get/77683/157551
10. https://www.technobuffalo.com/here-are-all-the-smart-email-assistants-you-can-use-to-hit-inbox-zero
11. https://smartassist.io/
12. https://www.nylas.com/blog/email-parsing-creating-structure-to-unstructured-data/?gclid=CjwKCAjw5Ij2BRBdEiwA0Frc9d-sxXbJ1GngZErpUJMo4GovgEgsyiOP1pma0ci1AD6YHvCAAaYGyBoCy8YQAvD_BwE