Visit Logger

High Level Solution Design Document

Author: Gabriel Accad

Document History

Revision History

|  |  |
| --- | --- |
| Date of this revision: 21/06/2021 | Date of next revision *(date)* |

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Revision Date | Summary of Changes | Changes marked |
| 1 | 21/06/2021 | Initial Draft | (N) |
| 2 | 06/07/2021 | Added Cognito Authentication  Changes front end to React framework | (N) |

Contents

1. Preface 5

1.1 Introduction & Background 5

1.2 Purpose of this Document 5

1.3 Scope 5

1.3.1 Included In Scope 5

1.3.2 Excluded from Scope 5

2. Architecture Overview 6

2.1 Overall Architectural Decisions 8

2.2 Database Structure 8

2.3 Serverless Functions Details 8

2.4 Flow Descriptions 9

2.4.1 Visit Logging 9

2.4.2 Email Notification 9

2.4.3 Visit Data Retrieval (Web) 10

2.4.4 Legacy Data Loading 10

2.4.5 Visit Data Analysis 10

3. Project Evolution 11

3.1 Summary of Project Phases 11

3.1.1 Core Functionality 11

3.1.2 Email Notifications 12

3.1.3 Reporting Solution 13

3.1.4 Legacy Data Loader 14

# Preface

The present document describes the design of the Visit Logger application

## Introduction & Background

The company ERP lacks the capability to record salesperson visit data to customers. The sales executives need to complete 5 or more visits per day and cover all their customers during the month. This lack of functionality makes it very difficult to track their performance as it is currently being done manually.

Sales managers have no way of analyzing the data or be notified when an important customer has been visited. This is affecting the sales performance of the company and its revenue potential.

The Visit Logger application proposed a solution to the above problem by offering a simple web interface for the sales people to enter their customer visits at the end of the day or while they are on the move.

The sales managers are notified when a VIP customer is visited and have access to an analysis platform to run their reports.

## Purpose of this Document

This document describes the design and architecture of the solution and presents the basic technologies used.

## Scope

### Included In Scope

The scope of the solution includes:

* High Level Design overview of the solution
* Description of each function
* Phased approach of the project
* Link to Github repository: <https://github.com/gaccad/VisitLogger>
* Link to working application website: <https://dev9334.d137fzgar94quk.amplifyapp.com/>

### Excluded from Scope

* Details of code use in lambda function and JavaScript frontend (can be found on the github repository).

# Architecture Overview

The Visit Logger application consists of multiple components that provide individual capabilities that address specific aspects of the overall solution. These components include:

* Hosting of frontend dynamic website with JavaScript framework
* Internal repository for code maintenance and commits
* API Gateway solution to expose REST API function of backend logic
* Serverless execution of code required for the 5 different functions of the application
* NoSQL backend database to store visit data recorded by users as well as legacy data from previous system
* Email notification functionality to notify supervisors when a visit to a VIP customer is logged
* Object storage containing Legacy Data to be loaded via serverless function to the NoSQL DB
* Reporting and analysis solution to aggregate and present data to management

The below picture gives a high-level view of the architecture of the Visit Logger solution. The required capabilities are mapped to a set of AWS products that provide the required functionality.

Each of the components is listed below and will be further detailed in the following chapters.

|  |  |
| --- | --- |
| AWS Amplify |  |
| AWS Lambda |  |
| API Gateway |  |
| AWS DynamoDB |  |
| AWS CodeCommit |  |
| AWS Simple Email Service |  |
| AWS Athena |  |
| AWS Quicksight |  |
| AWS S3 |  |
| AWS Cognito |  |
| AWS IAM |  |



***Architecture Overview of the Visit Logger Application***

## Overall Architectural Decisions

* All functionalities should be serverless whenever possible
* NoSQL to be used instead of RDBMS for cost efficiency and due to the loose schema structure
* Python to be used as language for Lambda functions
* All functions to run on roles based on least required privileges principle
* Internal code to be stored on AWS CodeCommit and mirrored on Github for public access
* AWS QuickSight to be used as analysis solution thanks to the recent DynamoDB connector availability
* SPICE to be used for Quicksight analysis retrieval: Real time data is not required and large dataset is difficult to manipulate real time
* DynamoDB insert trigger to be used for email notification instead of adding code to RecordVisit function: decoupling of functionality and scalability options for future event driven calls

## Database Structure

* ID (Primary Key)
* Customer Name
* Salesperson (Sort Key)
* Date
* Meeting Type
* Meeting Subject
* Meeting Details
* VIPCustomer
* Feedback
* LegacyData

## Serverless Functions Details

* RecordVisits
  + Function: Received JSON object as a parameter from index.html and parses it then puts data into VisitTable in DynamoDB
  + Trigger: REST API
* GetVisits
  + Function: Retrieves visit data from VisitTable and passes it to index.html via a JSON object
  + Trigger: REST API
* EmailSupervisor
  + Function: When a VIP customer visit is logged, the function receives the NewImage JSON object from DynamoDB and sends an email to the supervisor with visit details
  + Trigger: DynamoDB Stream
* LegacyDataLoader
  + Function: Reads csv file from S3 object storage, parses it and loads the data into the VisitTable in DynamoDB. Only used once to load legacy data.
  + Trigger: Manual
* AthenaDynamoDBConnector
  + Function: In built function that connects DynamoDB VisitTable to Athena which is in turned used by QuickSight for data aggregation and analysis
  + Trigger: QuickSight query

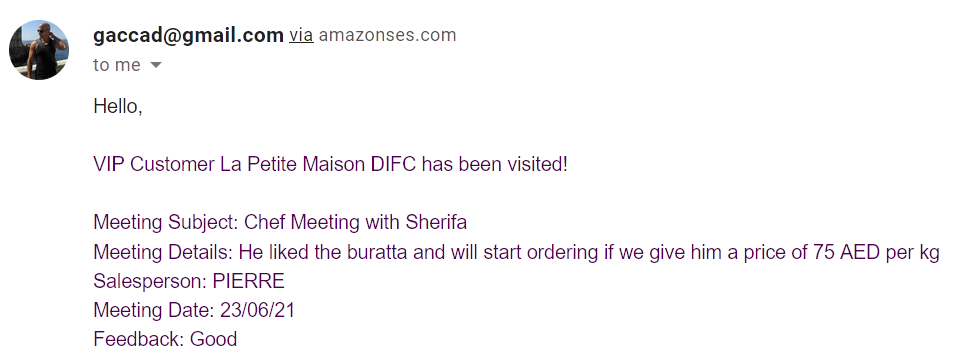
## Flow Descriptions

### Visit Logging

1. User opens the webpage and enters the required data
2. The page calls the REST API and passed the captured parameters
3. REST API calls the lambda function RecordVisit
4. The lambda function puts the data into the DynamoDB table VisitTable
5. The user is notified via popup that the visit logging was successful

### Email Notification

1. User enters visit data and set the ‘VIP Customer’ field to ‘Yes’
2. DynamoDB event stream is triggered on insert and calls the EmailSupervisor lambda function
3. The function uses SES to send email to the supervisor (currently hardcoded) with visit details



***Example of email notification***

### Visit Data Retrieval (Web)

*Functionality is currently hidden. Can be tested from get.html page*

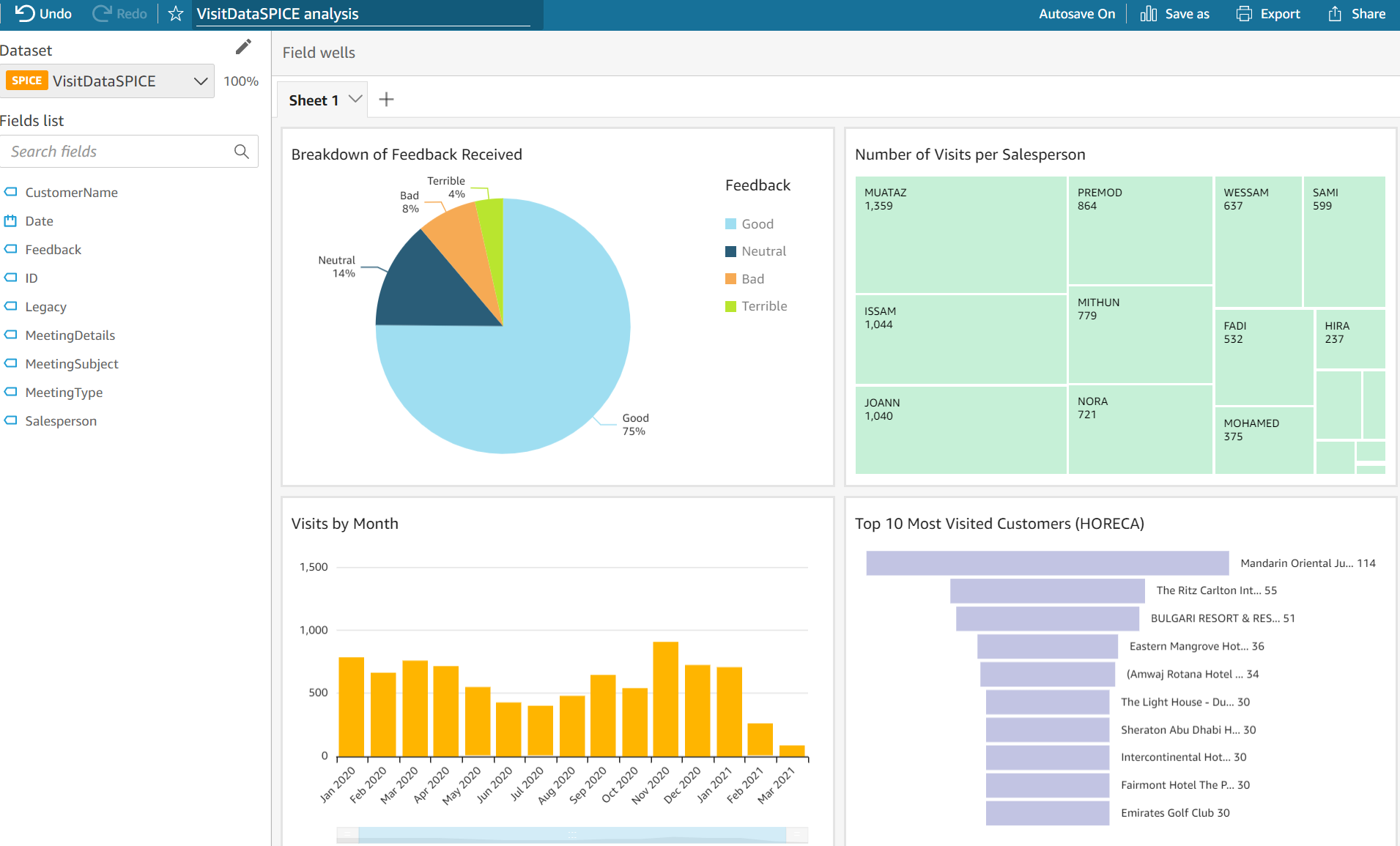
1. User clicks on ‘Get recently entered visits’ link
2. Function returns predefined number of visits

### Legacy Data Loading

1. Legacy data is loaded into S3 in csv format
2. Disable EmailSupervisor Lambda trigger to avoid having to call the function on every insert
3. Increase timeout on lambda function from 3 seconds to 180 seconds
4. Lambda function LoadLegacyData is manually executed to load the data into DynamoDB
5. Enable EmailSupervisor Lambda trigger

### Visit Data Analysis

1. Sales manager logs into the Quicksight portal using his username and password
2. User selects a pre-defined analysis dashboard or can create a new one



***Pre-defined dashboard using SPICE Dataset***

# Project Evolution

## Summary of Project Phases

### Core Functionality

Base three tier architecture to record and display visits:

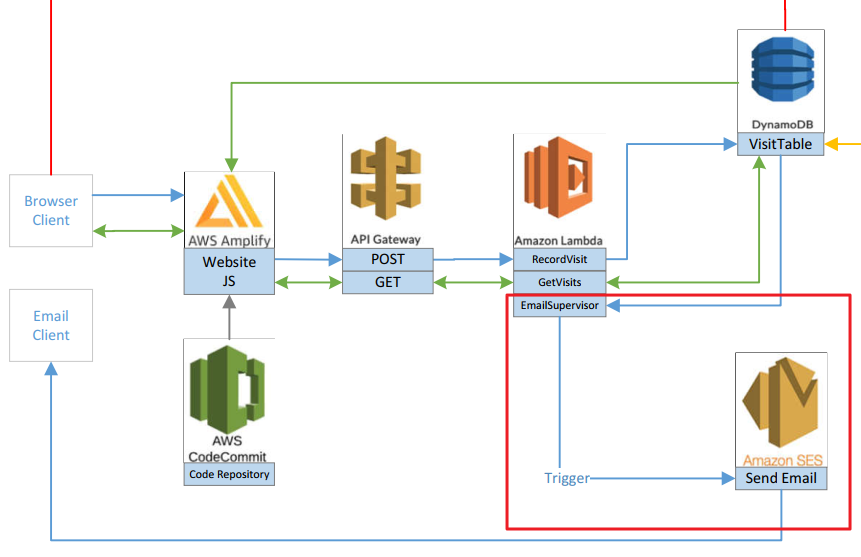
* Frontend form using JavaScript to call the REST API
* API Gateway has 2 methods which call their respective Lambda functions
* Lambda RecordVisit (described in section 2.4.1)
* Lambda GetVisits (described in section 2.4.3)
* DynamoDB Table
  + ID as primary key
  + Salesperson as sort key



### Email Notifications

Added functionality to email supervisor on visit to a VIP customer

* Lambda function with checks for ‘CustomerVIP’ flag == ‘Y’ and calls SES to send an email with meeting details
* DynamoDB trigger which calls the above function on insert



### Reporting Solution

The goal was to pull the data into Quicksight for Analysis, the new DynamoDB connector requires the following which were added:

* Athena Catalog
* Lambda function which acts as a connector between DynamoDB and Athena
* S3 bucket to store spill data from the Athena Catalog
* Quicksight subscription
  + New dataset created in SPICE mode
  + New dashboard created with frequently used metrics



### Legacy Data Loader

The data from the previous system (8571 records) was loaded into DynamoDB to be used as part of the trend analysis. DynamoDB has no current functionality to upload data from a csv file, hence the below functionality was added:

* S3 bucket to store the csv file
* Lambda function was written to iterate through the rows and put them into the VisitTable in DynamoDB

### 

### User Authentication

The user is presented with a sign in page and has the following options:

* Create a set of credentials which will be added to the user pool. User will receive an email validation code and can log in once validated.
* User can log in with existing credentials



