California State University, San Bernardino
CSUSB ScholarWorks

Electronic Theses, Projects, and Dissertations | Office of Graduate Studies

5-2023

GOVERNMENT AID PORTAL

Darshan Togadiya

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd

 Part of the Computer and Systems Architecture Commons

Recommended Citation

Togadiya, Darshan, "GOVERNMENT AID PORTAL" (2023). *Electronic Theses, Projects, and Dissertations*. 1753.
https://scholarworks.lib.csusb.edu/etd/1753

This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

GOVERNMENT AID PORTAL

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfilment

Of the Requirements for the Degree

Masters of Science

In

Computer Science

by

Darshan Jitendrabhai Togadiya

May 2023

GOVERNMENT AID PORTAL

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

Darshan Jitendrabhai Togadiya

May 2023

Approved by:

Amir Ghasemkhani, Advisor, School of Computer Science and Engineering

Khalil Dajani, Committee Member

Jennifer Jin, Committee Member

© 2023 Darshan Jitendrabhai Togadiya

ABSTRACT

In today's world, contacting government officials seems a big task when it comes to reporting small concerns. There are many authorities and officials which makes it very difficult for ordinary people to figure out who they should contact to resolve their daily issues. To address this problem, I have developed an application which can act as intermediary between citizens and government authorities. This portal will enable locals to submit complaints regarding personal or general issues through a complaint form, which will then be routed to the appropriate government department. Once a complaint is filed, government teams are immediately alerted and can use the program to gain more insight into the issue. Furthermore, the app will notify the complainant about the status of their request, providing updates on whether the problem has been resolved. The app also includes a feature that allows users to provide detailed information about a problem by selecting a specific problem category, location, and uploading relevant photos. This will give government officials a better understanding of the issue and allow them to take immediate action. To build this portal, I leverage Amazon web service such as Amazon Lambda, Amazon S3, Amazon DynamoDB, Elastic Beanstalk, AWS Lambda, AWS Cognito, API Gateways, and Secret Manager. Furthermore, this application will contain all the mentioned features along with user-friendly interface to provide more convenient experience. The application will be available for use

as a software and hence, the delivery model will be Software as a Service (SaaS). The users will use the services directly, and they will not require access to the infrastructure components or the platform on which the application will be built.

ACKNOWLEDGEMENTS

I am writing to express my gratitude to Dr. Amir Ghasemkhani, my Committee Chair, for his continuous support and for pushing the boundaries each time to create this research work as the best version. Moreover, I want to thank my committee members, Dr. Khalil Dajani and Dr. Jennifer Jin, for believing in me and agreeing to join the committee. I am grateful for their trust in me while working on this project. I'd also like to express my gratitude to all the University's professors for assisting me in getting to this point in my academic career.

I am thankful that the School of Computer Science at California State University San Bernardino has modeled a curriculum to help me achieve my future goals and endeavors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

PROJECT OVERVIEW

Introduction

Governments around the world are actively working to enhance the quality of life for their residents. They provide better infrastructure, affordable education and a safe and secure living environment. Yet despite all their efforts, many people still face personal or general issues that may not be addressed effectively due to an extensive process when contacting government authorities. Furthermore, lack of awareness about reporting procedures often discourages people from raising concerns which leaves many issues unresolved.

To address these challenges, I am proposing the creation of a public aid portal that will offer an accessible and dependable means for people to voice their complaints to higher authorities. This platform will enable users to submit complaints about general matters via an online complaint form which will then be processed and forwarded on to government departments.

Statement of Problem

Reporting issues and problems to authorities can be a time-consuming and laborious process, and many may not know how to go about it. As a result, many issues remain ignored by higher authorities; thus, there is a need for an easy platform that allows citizens to file complaints about personal or general problems and notifies government departments so they can take appropriate actions in response.

Purpose of the study

To satisfy this demand, a public help application is developed where residents can raise their concerns and send them directly to the responsible authorities. This application allows users to file a complaint which can be anything related to general or personal issues. To file a complaint, user will have to fill a form and give detailed information regarding their issue. Authorities will be notified whenever someone register an issue. Moreover, the app keeps a track of the status of complaint which is submitted by any user. There is an option of uploading photograph, giving location details and categories of complaint like cybercrime,

road construction issue, etc. This will help the government to have the better understanding about the problem.

Definitions

The government aid portal is a software-as-a-service which is built for residents where anyone can file a complaint in order to live in a good environment. Generally, to file a complaint there is very tedious process which everyone has to follow. By using this app, no one has to go through any lengthy process even just to raise a small issue. This application is connected to Amazon Web Service which is also used to store the data and notify about the complaint status.

Features of Public Aid Portal:

Complaint Form: Residents can fill out and submit their grievances using the portal.

Complaints can be categorized on the portal based on issue kind, location, and urgency.

File Attachments: Users can attach images and other detailed documents to the portal to help government officials.

Tracking of Complaints: The platform offers a tool that tells complainants about the status of their complaint requests.

Amazon Web Services Integration: The gateway will be integrated with several Amazon Web Services, including Amazon Lambda, S3, DynamoDB, Elastic Beanstalk, AWS Lambda, Cognito, API Gateways, and Secret Manager.

CHAPTER TWO

LITERATURE REVIEW

Current State of Art

In recent years, practitioners have become interested in complaint management systems. Several studies on their development and implementation for governments have been undertaken. In China, Song and Li (2019) suggested a complaint management system that made use of both mobile applications and cloud-based platforms. This allowed users to lodge complaints swiftly while obtaining real-time updates on the status of their complaints. This enhanced efficiency throughout the complaint-handling process, as demonstrated by these authors, as well as communication between users and government agencies.

Al-Mashari and Zairi (2000) created a complaint management system for a Saudi government organization. This includes an electronic complaint form, automatic tracking, and in-depth analysis. According to the authors, this improved service quality provided by the organization, reduced response times to grievances and boosted customer satisfaction levels.

Studies indicate a growing interest in developing complaint management systems for governments worldwide. These solutions have the potential to improve service delivery and citizen communication. However, the research also identifies significant gaps and limitations that must be addressed.

Gaps and Limitations

Though interest in government complaint management systems has expanded, there are still gaps and limitations in the existing research. One noteworthy restriction is the scarcity of studies integrating complaint management systems with cloud-based services such as Amazon Web Services (AWS). The proposed project aims to fill this need by integrating the Public Aid Portal to AWS services such as Lambda, S3, DynamoDB, and others. This integration will establish an effective platform for complaint management while boosting reliability at all levels of complaint processing.

Another disadvantage of the dearth of research on complaint management systems in developing countries is the difficulty in comprehending the specific cultural, social, and economic factors that may affect the adoption and efficacy of these systems. Developing countries often have different governance structures,

regulatory environments, and societal norms that may influence the way complaints are made and addressed. Most studies have focused on developed nations, with little research into the challenges and opportunities related to implementation in less developed nations. Most studies have focused on developed nations, with little research into the challenges and opportunities related to implementation in less developed nations. Thus, this project seeks to fill this gap by providing insights into these countries' experiences with complaint management system implementation. This will contribute significantly to the existing literature by providing an understanding of these obstacles and opportunities.

Therefore, the project will contribute to the literature by providing insights into the challenges and opportunities of implementing complaint management systems in developing countries.

Relevance of the Project

The literature review indicates the proposed project is both pertinent and timely, given the growing demand for complaint management systems for governments worldwide. To address gaps and limitations in existing literature, this proposal integrates Public Aid Portal with AWS services while targeting developing countries. Having a reliable platform with AWS services will offer insights into

challenges associated with implementing such systems in these nations, while targeting developing ones will offer insights into potential solutions they could face when doing so.

The literature review also provides insights into the best practices and lessons learned from similar projects worldwide, which can be applied to the proposed project. For instance, the study by Song and Li (2019) highlights the importance of user-friendliness and real-time updates in complaint management systems. These insights can be used to design the user interface and the notification system in the Public Aid Portal to enhance the user experience and improve the efficiency of the complaint management process.

CHAPTER THREE

REQUIREMENT ANALYSIS

Functional Requirements

There are some following user requirements which are essential to be implement in this project:

1. Sign-up Feature:
    - Users should be able to sign up for the application by providing personal information and credentials.
    - User data should be collected and stored in the AWS Cognito User Pool.

2. Complaint Form:
    - Users should be able to submit their complaints through a complaint form.

- The complaint form should collect pertinent information such as the subject, category, location, description and photos related to the grievance.
- All complaint-related data should be stored in DynamoDB database.
- Additionally, an access link for images stored in S3 bucket should also be stored within the database.

3. Notification Feature:
    - Users should receive email notifications regarding the progress of their complaints.
    - These should include details such as the complaint's status and estimated time for resolution.

4. Mobile and Web Applications:
    - The application should provide a mobile application for general users and web application for administrators.
    - Users should have access to both applications, depending on their user type.

5. Scalability and Availability:

- The application must be scalable and capable of handling traffic dynamically while still retaining performance.

Non-functional Requirements

There are some following non-functional requirements for this project:

1. User-friendly Interface:
    - The application should provide a user-friendly interface that displays all elements clearly and efficiently.
    - The interface should be built using React Native and ReactJS.

2. Backend Development:
    - For backend development, the application should utilize NodeJS and Express.

3. Security and Data Integrity:
    - For security and data integrity, the application should be secured using AWS Elastic Beanstalk, Amazon Secret Manager, and Amazon SNS.

4. Reliability and Fault-tolerance:
    - The application must be highly reliable and resilient, using AWS services for auto-load sharing.

Constraints and Assumptions

1. Frontend and Backend Development:
    - According to my skillset, the application will utilize React Native and ReactJS for the frontend, along with NodeJS and Express in the backend.

2. Data Storage and Retrieval:
    - As part of this project, AWS Elastic Beanstalk, Amazon Secret Manager, Amazon SNS and DynamoDB will be utilized for data storage and retrieval.

3. Auto-load Sharing:
    - This application will utilize AWS services for auto-load sharing to efficiently manage traffic.

4. Lambda Function:
    - The application will utilize a Lambda function to convert Base64 objects to image objects and store them in an S3 bucket, along with complaint-related data stored in DynamoDB database.

Use case scenarios

1. User submits a complaint:
    - Downloading the mobile application requires users to sign in using their credentials, then submit a complaint form by filling in all required information and uploading images.

2. Admin views complaints:
    - An admin can log into the web application and view complaints submitted by users, along with their related data and uploaded images.

3. User receives a notification:
    - Users receive an email notification regarding the status of their complaint, which includes details such as its current state and estimated time for resolution.

CHAPTER FOUR

DESIGN AND ARCHITECTURE

Design Principles and Patterns

The design of the system is crucial for any software project and should adhere to certain principles and patterns for maximum efficiency and scalability. One such principle is the separation of concerns, which means designing the system so that different aspects are separated - such as user interface logic from application logic; data storage from user interface logic - in order to facilitate easier upkeep or modification of the system. System design typically segregates different concerns into distinct modules or components. For instance, the user interface is separated from application logic and data storage.

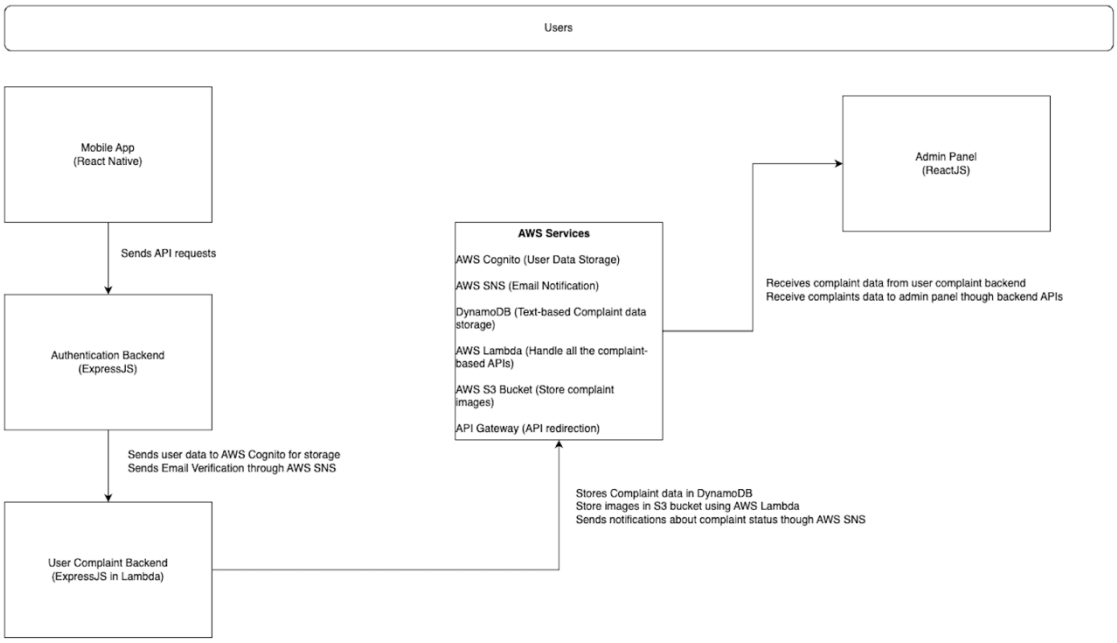

Figure 1: Block Diagram of the system

The user interface is implemented using React Native and ReactJS (Mobile App and Admin Panel component), while the application logic is handled through microservices such as the User Management Service, Complaint Service, and Notification Service (AWS Services component). However, there is two separate backend code that is responsible for handling the user requests and perform necessary operations (Authentication Backend component for handling user authentications and User Complaint Backend for handing all the complaint related actions).

Data storage and retrieval are handled through AWS DynamoDB, while images related to complaints are stored using AWS S3. Each component has its own distinct responsibility that is clearly separated from others; for instance, the Complaint Service receives, stores and retrieves complaints through integrations with AWS DynamoDB for data storage and retrieval as well as AWS S3 for complaint-related images.

By breaking up these concerns into distinct components, each can be designed, tested, and maintained independently. Not only does this simplify the system but also increases its efficiency and scalability.

Another crucial principle is the Single Responsibility Principle (SRP), which states that each module or component should have a single responsibility. This helps keep code modular and easy to maintain and modify. Moreover, Model-View-Controller (MVC) pattern should also be utilized when designing the system; it divides an application into three components: model (data and business logic), view (user interface), and controller (handles user input). Utilizing this pattern helps structure code so it becomes simpler to manage over time.

Here is an overview of how the MVC pattern is utilized in our project:

1. Model: The data and business logic for your system is captured in the data models and database design section of your project. We utilize AWS DynamoDB for data storage and retrieval, with two tables: a user table for user information and a Complaint table storing complaint-related details. Moreover, authentication setup is included as part of this model - Express app code creating Cognito users and managing authentication controls.

2. View: View is responsible for displaying the user interface of an application. In this project, it was developed with React Native and ReactJS, featuring components like sign-up forms, complaints list, and notification centers.

3. Controller: The application logic of the system is handled through microservices architecture, which consists of several components described in this project's system architecture and components section. Each service has its own controller which takes care of user input, processes requests, and communicates with other services as required. Moreover, Express app code for authentication also contains its own controller for managing user authentication and authorization.

The MVC pattern assists in decoupling the various concerns of an application, making it simpler to manage and modify. Furthermore, it facilitates parallel development by different teams working on distinct components of the same program.

Loose coupling is another important design principle to adhere to. This implies that system components should be loosely coupled in order to reduce their dependencies on one another, making for simpler maintenance and modification of the system.

For this project, loose coupling is achieved through the use of microservices architecture. The different services in the system are independent and communicate with one another via APIs, thus reducing their dependencies between them. For example, the Complaint Service is responsible for receiving, storing and retrieving complaints; it communicates with the User Management Service for user authentication and authorization. These two services are loosely coupled and can be developed and deployed independently. Likewise, the Notification Service sends email notifications to users about their complaints' statuses without direct dependencies on either Complaint Service or User Management Service.

Furthermore, AWS services like AWS DynamoDB, S3, and SNS encourage loose coupling by offering scalable and reliable data storage, retrieval, and communication that can be accessed independently by different components of the system. This guarantees that changes made to one component do not impact other functions; each can be independently designed, deployed, and scaled independently.

System Architecture and Concepts

A microservices architecture should be employed when designing the system to guarantee its scalability and modularity. This architecture consists of small, independent services that communicate with one another via APIs. This makes for easier maintenance and modification to the system as well as improved scalability and fault tolerance.

The system should consist of three components: a User Management Service, Complaint Service and Notification Service. The User Management Service should manage user authentication and authorization through AWS Cognito User Pool integration. The Complaint Service handles complaints receiving, storing and retrieving them via AWS DynamoDB data storage/retrieval

as well as AWS S3 for complaint-related images storage. Lastly, the Notification Service sends email notifications to users regarding their complaints' status via AWS SNS protocol.

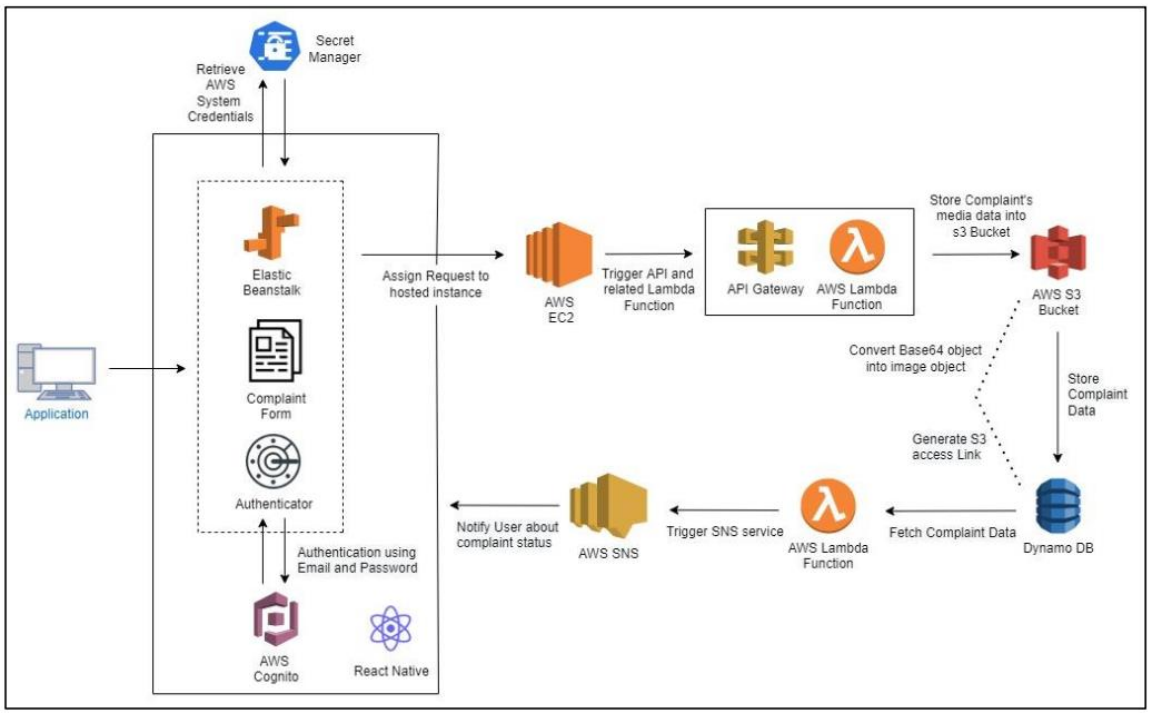

Figure 2: Final architecture of application

Data Models and Database Design

For data storage and retrieval, the system should utilize AWS DynamoDB. DynamoDB is a highly scalable NoSQL database that can handle large amounts

of information with speedy access. For user authentication, system uses AWS Cognito pool to store user information in a pool which will later be utilized by other services in AWS for this project.

DynamoDB will generate one tables and Cognito will create a pool: The Complaint table holds complaint-related details.

The User table should contain the following attributes:

1. First name: The first name of the user.
2. Last name: The last name of the user.
3. Email: The email address of the user, which will be used for communication purposes.
4. Mobile number: The mobile number of the user.
5. Password: The encrypted password of the user, which should be stored securely.
6. Citizenship: The citizenship of the user.
7. Gender: The gender of the user.

The Complaint table should contain the following attributes:

1. Complaint ID: A unique identifier for each complaint in the system.
2. Subject: A brief description of the complaint.
3. Severity: The severity level of the complaint, such as low, medium, or high.
4. Category: The category of the complaint, such as infrastructure, safety, or sanitation.
5. Username: The username of the user who submitted the complaint.
6. Complaint status: The status of the complaint, such as open or closed.
7. User email: The email address of the user who submitted the complaint.
8. Photos: The processed image related to the complaint.
9. Description: A detailed description of the complaint.
10. Location: The location where the complaint occurred.

It is essential to select a primary key carefully for each table in order to guarantee optimal performance and efficient data retrieval. In my case, I used the complaint ID as the primary key in the Complaint table to facilitate quick retrieval of individual complaints.

In addition to creating tables, the system must ensure proper indexing and partitioning are in place to optimize query performance. Furthermore, data validation must be implemented and ensure consistency and integrity across the system. Overall, data models and database design are essential elements of your project; you should take great care to guarantee they meet all system requirements and scale efficiently with its expansion.

User Interface Design

The system should provide a user-friendly interface that displays all elements seamlessly, using React Native and ReactJS. React Native is a framework designed specifically for mobile app development using the ReactJS library. Sign-up forms, complaint forms, complaint lists, and notification centers should all be part of the interface.

User information, such as personal details and credentials, should be collected via the sign-up form. Users should fill out the complaint form with information such as the issue, category, location, description, and images. A complaints list should display all user-submitted concerns, and a notification center should offer status updates.

Security and Privacy Consideration

Security and privacy must be considered while developing any system, particularly one that manages user data. As a result, this project should be planned with these concepts in mind.

Secure user authentication and authorisation is one of the most important security procedures that must be put in place. To do this, AWS Cognito User Pool, a fully managed user directory service that provides secure user authentication and authorisation via industry-standard protocols such as OAuth 2.0 and OpenID Connect, should be used.

Cognito User Pool by AWS enables the system to deliver secure user authentication and authorization without the need to operate a proprietary authentication and authorization solution. In addition, Cognito User Pool offers features like multi-factor authentication, password policies, and integration with other AWS services.

Data encryption is another essential security measure that should be considered. The system must guarantee all sensitive data, like passwords and

user info, is encrypted both at rest and during transit. This can be achieved using AWS KMS for managing encryption keys and AWS SSL certificates to encrypt data during transmission.

Privacy considerations should also be taken into account when designing the system. It must abide by all relevant privacy regulations, such as GDPR, and guarantee that user data is collected and used only for legitimate purposes. Furthermore, users should have transparency and control over their data through features like viewing and deleting it.

Overall, designing the system with security and privacy in mind is essential to protect user data and maintain user confidence in the system.

CHAPTER FIVE

IMPLEMENTATION AND TESTING

Technology Stack

The frontend of the application is divided into two parts. The mobile application is built for the general people and admin will use web application. So, the frontend of our application is built using React Native and ReactJS [4]. While the backend of the application is built using NodeJS and Express [5]. I deployed both these frontend and backend to Elastic Beanstalk using YAML file for deployment of cloud formation. All the major APIs of the application related to the people's complaints are deployed to Lambda function and are accessible through generated serverless APIs.

Project features category and services:

Table 1: Decided AWS service from each category

Category	**Services**
Computer	1. AWS Elastic Beanstalk 2

	2. AWS Lambda
Storage	1. AWS S3 2. AWS DynamoDB
Network	1. AWS API Gateway
General	1. Amazon Cognito 2. AWS SNS 3. AWS Secret Manager

Testing Methodologies

The project employed various testing methodologies to guarantee the quality of its application. Unit testing was carried out using Jest testing framework for backend APIs; integration testing was done via Postman tool for APIs and frontend user interface; end-to-end testing was conducted using Cypress framework for overall functionality assessment.

Unit testing:

Unit testing is a software testing methodology where individual units of code are tested in isolation to verify that each unit works as expected.

Introduction

The aim of this testing is to identify and fix errors in the backend APIs, and ensure that the application meets the requirements.

Testing Framework

The Jest testing framework was used for unit testing the backend APIs, as well as for testing the frontend functionalities of the React Native app and the admin portal built with ReactJS.

Test Cases

NodeJS App (Backend)

The following test cases were developed and executed for the backend APIs:

User Model

- Test to ensure that a new user can be created and saved to the database.
- Test to ensure that a user cannot be created with missing required fields.
- Test to ensure that a user can be retrieved from the database by their ID.
- Test to ensure that a user can be retrieved from the database by their username.
- Test to ensure that a user's details can be updated in the database.

Complaint Model

- Test to ensure that a new complaint can be created and saved to the database.
- Test to ensure that a complaint cannot be created with missing required fields.
- Test to ensure that a complaint can be retrieved from the database by its ID.
- Test to ensure that a user can retrieve all complaints in the database.
- Test to ensure that a user can retrieve their complaints from the database.

React Native App (User)

The following test cases were developed and executed for the React Native app:

Authentication

- Test to ensure that a user can login with correct credentials.
- Test to ensure that a user cannot login with incorrect credentials.
- Test to ensure that a user can logout successfully.

Complaint Submission

- Test to ensure that a user can submit a new complaint.
- Test to ensure that a user cannot submit a complaint with missing required fields.

Complaint Management

- Test to ensure that a user can view their own complaints.
- Test to ensure that a user can view the status of their complaints.
- Test to ensure that a user can delete their own complaints.

ReactJS App (Admin)

The following test cases were developed and executed for the ReactJS app:

Authentication

- Test to ensure that an admin can login with correct credentials.
- Test to ensure that an admin cannot login with incorrect credentials.
- Test to ensure that an admin can logout successfully.

Complaint Management

- Test to ensure that an admin can view all complaints in the database.

- Test to ensure that an admin can update the status of a complaint.
- Test to ensure that an admin can delete any complaint.

Test Results

All test cases were executed successfully, and no errors were found. The application met the expected standards of functionality and performance.

Integration testing:

Introduction

This document outlines the integration testing process for the project. The aim of this testing is to ensure that the various components of the application are working together correctly and meeting the requirements.

Testing Framework

Postman testing tool was used for integration testing the APIs and the frontend user interface.

Test Cases

The following test cases were developed and executed for the application:

APIs

- Test to ensure that a user can register for an account through the registration API.
- Test to ensure that a user can log in to their account through the login API.
- Test to ensure that a user can create a new complaint through the complaints API.
- Test to ensure that a user can retrieve their complaints through the complaints API.
- Test to ensure that a user can retrieve all complaints through the complaints API.
- Test to ensure that a user can update their profile information through the profile API.

User Interface

- Test to ensure that the user can register for an account through the registration page.
- Test to ensure that the user can log in to their account through the login page.
- Test to ensure that the user can create a new complaint through the complaints page.
- Test to ensure that the user can retrieve their complaints through the complaints page.
- Test to ensure that the user can retrieve all complaints through the complaints page.
- Test to ensure that the user can update their profile information through the profile page.

Test Results

All test cases were executed successfully, and no errors were found. The application met the expected standards of functionality and performance.

End-to-end testing:

Introduction

The aim of this testing is to ensure that the application is functioning correctly and meets the requirements.

Testing Framework

Cypress framework was used for end-to-end testing the application.

Test Cases

The following test cases were developed and executed for the application:

User Functionality

- Test to ensure that a user can create an account and log in.
- Test to ensure that a user can submit a complaint.
- Test to ensure that a user can view their complaints.
- Test to ensure that a user can update their profile.

Admin Functionality

- Test to ensure that an admin can log in.
- Test to ensure that an admin can view all complaints.

- Test to ensure that an admin can update the status of a complaint.

Third-party Integration

- Test to ensure that the map is displaying correctly on the complaint submission page.
- Test to ensure that images are uploaded and displayed correctly on the complaint details page.

Test Results

All test cases were executed successfully, and no errors were found. The application met the expected standards of functionality and performance.

Results

User login screen

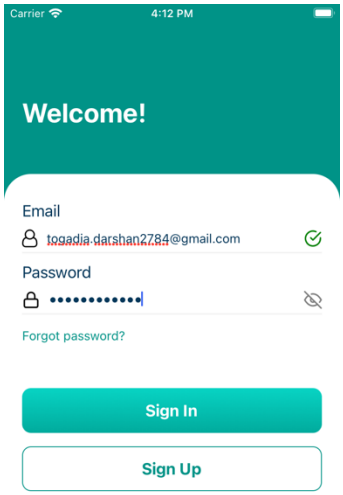

Figure 3: Login Screen

This is the login screen of mobile application. User has to login and authenticate themselves to access the functionality of the application. So, they can login with registered email address and password.

User registration screen

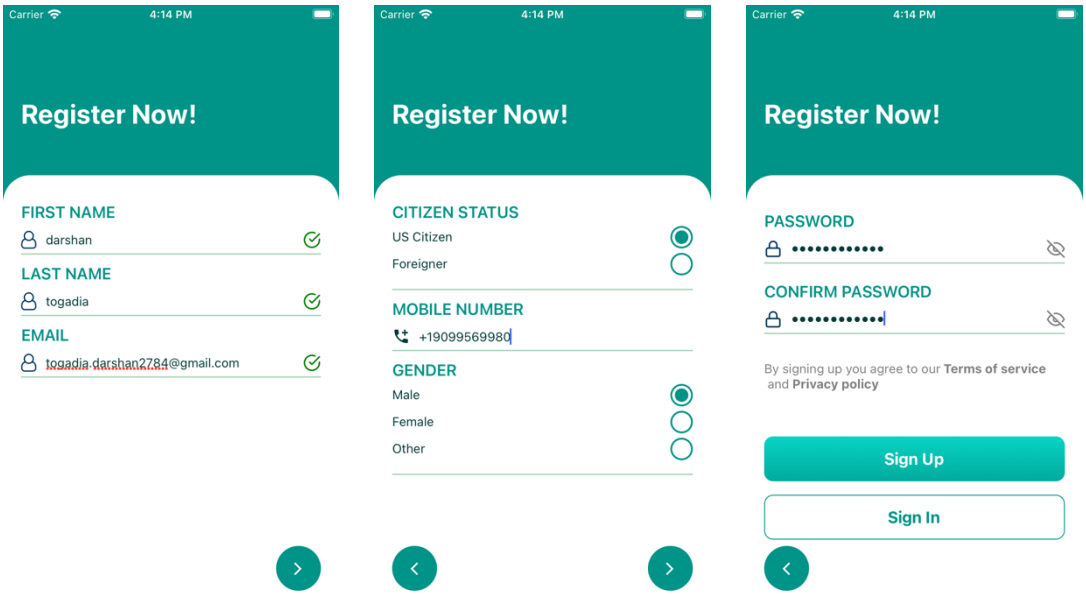

Figure 4: Registration Screens

This application requires users to fill in all fields listed on the form, then click "signup." The system will send two mails to that email address: one verifying their account and another subscribing them to our mail messaging service. After these

both tasks have been completed, users are registered and can utilize all features available within the application.

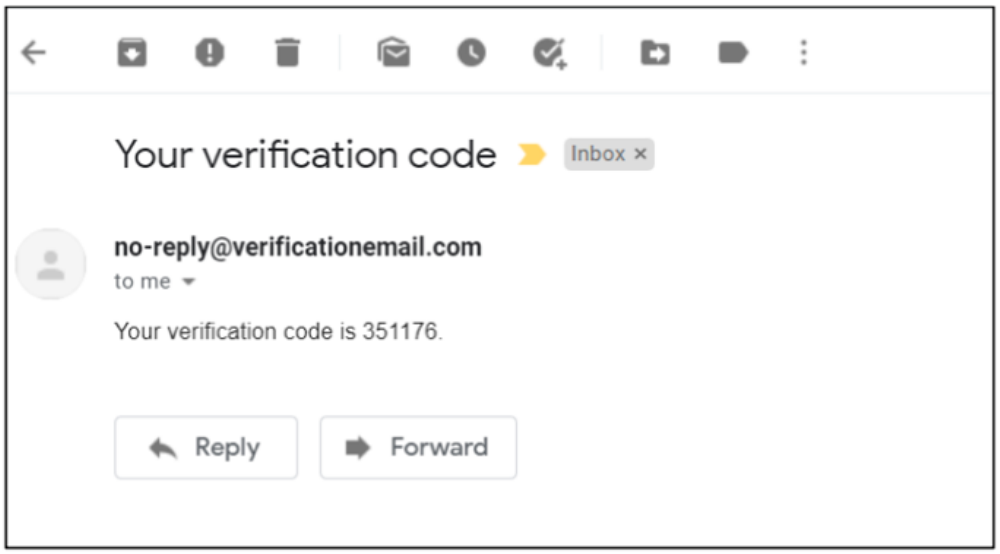

Figure 5: Email Notification to verify the user

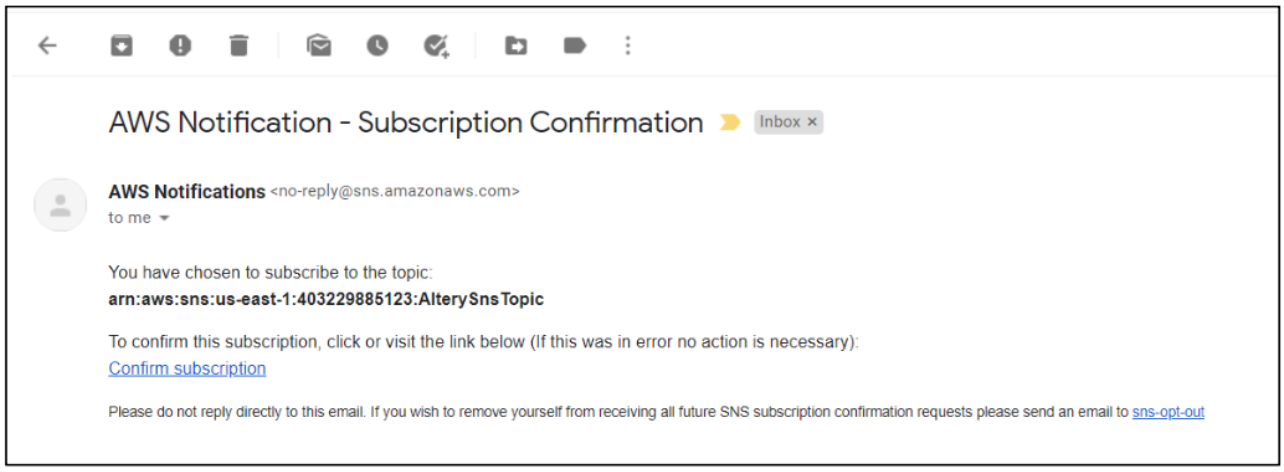

Figure 6: Email Notification to request for subscription

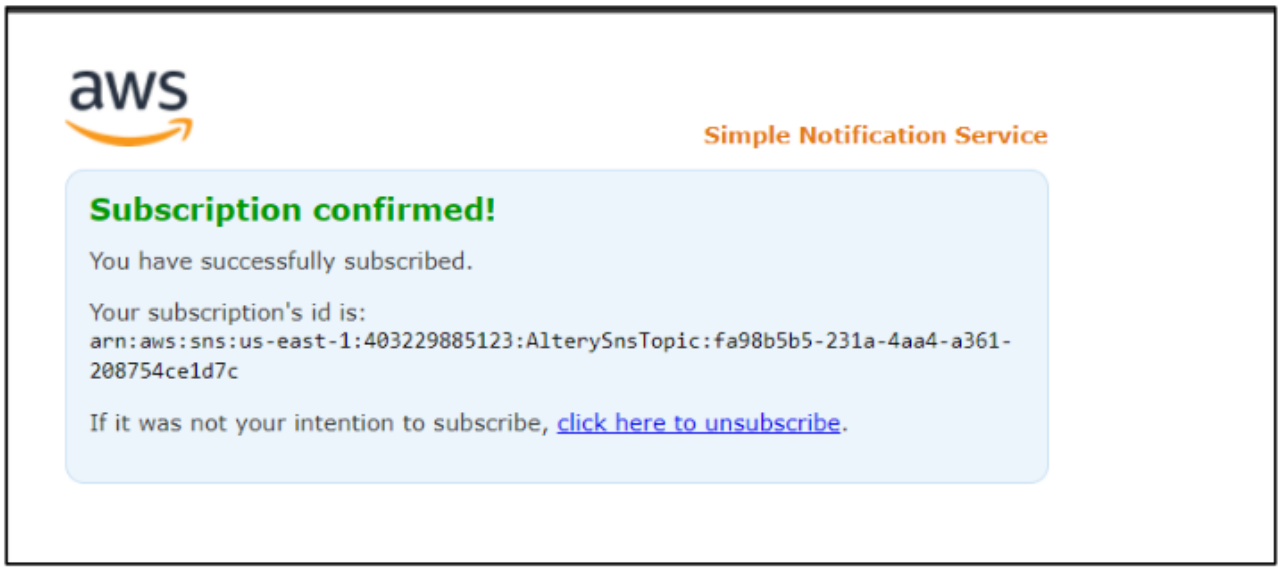

Figure 7: Screenshot of the subscription confirmation

User’s complaint dashboard, and details screen

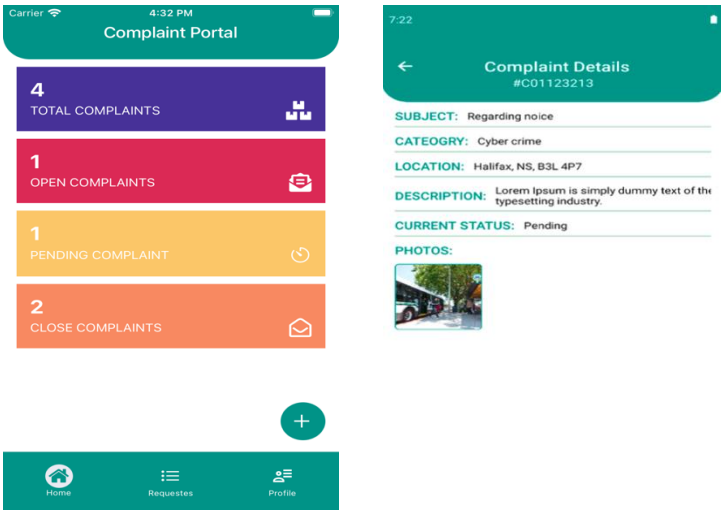

Figure 8: Complaints dashboard screen, and detail screen

This is the main screens of the application, where user can check all the complaints that they have made in the past, also they can check the complete details of the filled complaints.

Add new complaint screens

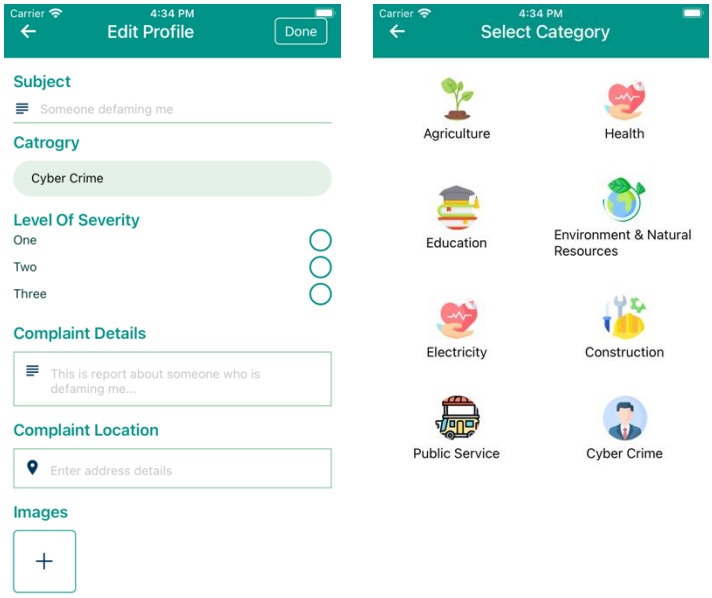

Figure 9: Add new complaint screen

Users can file a new complaint by filling this online form. They can open it by clicking on the plus button on their dashboard screen and select the category of complaint they wish to make, upload an image related to their problem, and click 'Done' when finished. Your request will then be sent directly to a government officer for review and resolution.

Admin login, dashboard page, and complaint details page

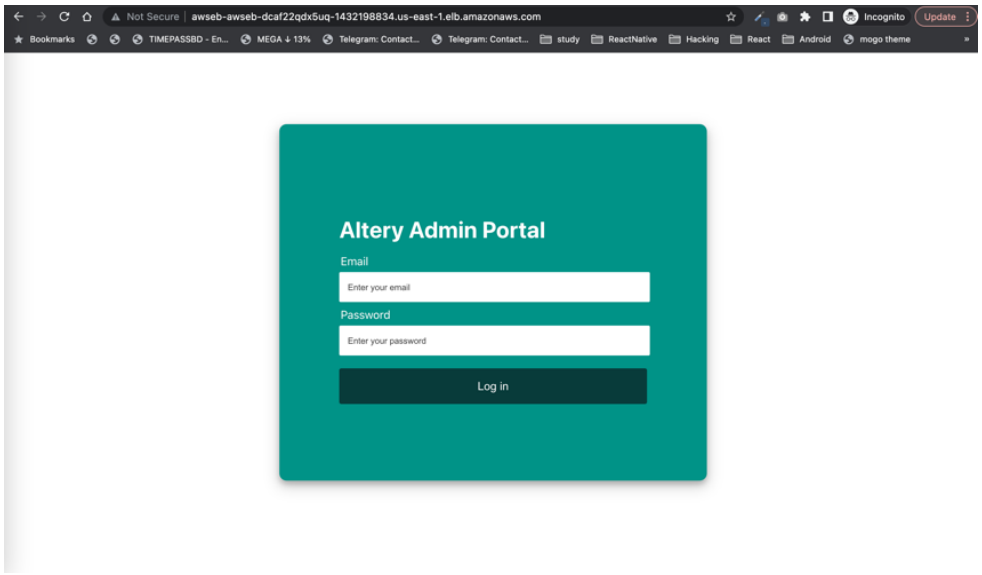

Figure 10: Admin login page

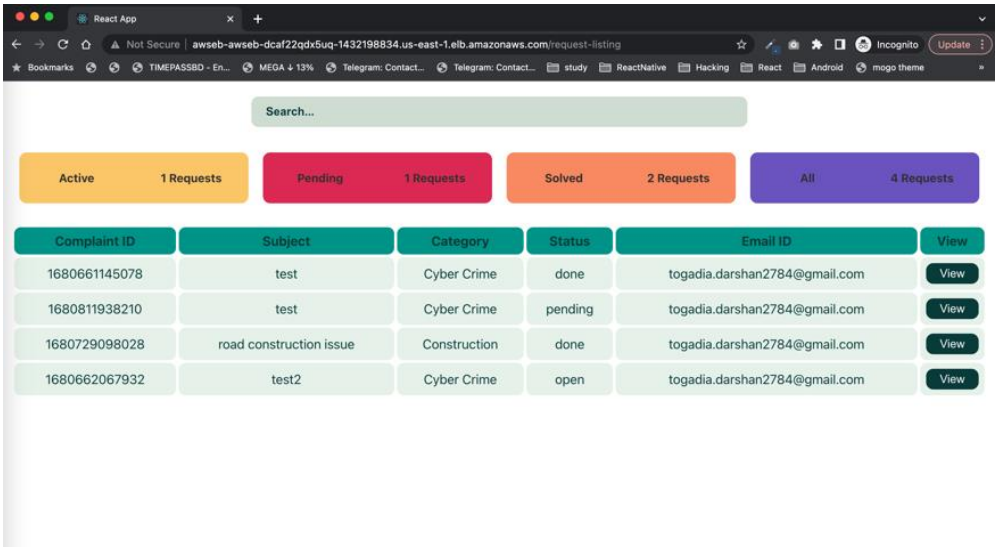

Figure 11: Admin complaint dashboard page

These screenshots depict an admin hosted web application. Figure 6 depicts the login page, while Figure 7 showcases the dashboard page of the application. Once an administrator logs in, they can access all requests made by people and check all major details like subject, category and complaint status by clicking a view button. Alternatively, they can view complete complaints by clicking "submit" button.

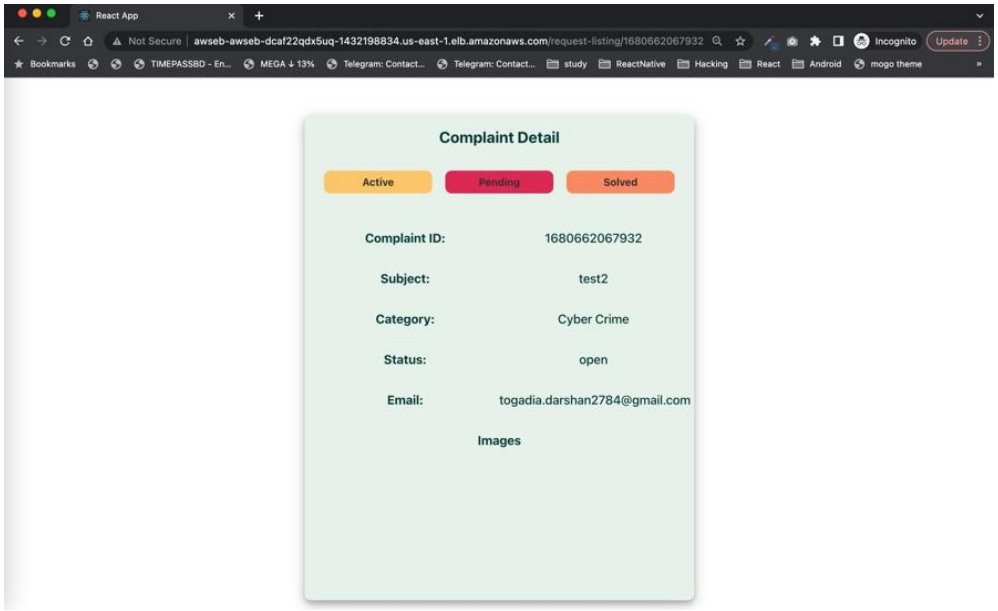

Figure 12: Complaint's detail page

Government officers can verify all details from this complaint details page and change the complaint status to active, pending or solved. Once they do this, the person who filed this complaint will be notified via email with its new status.

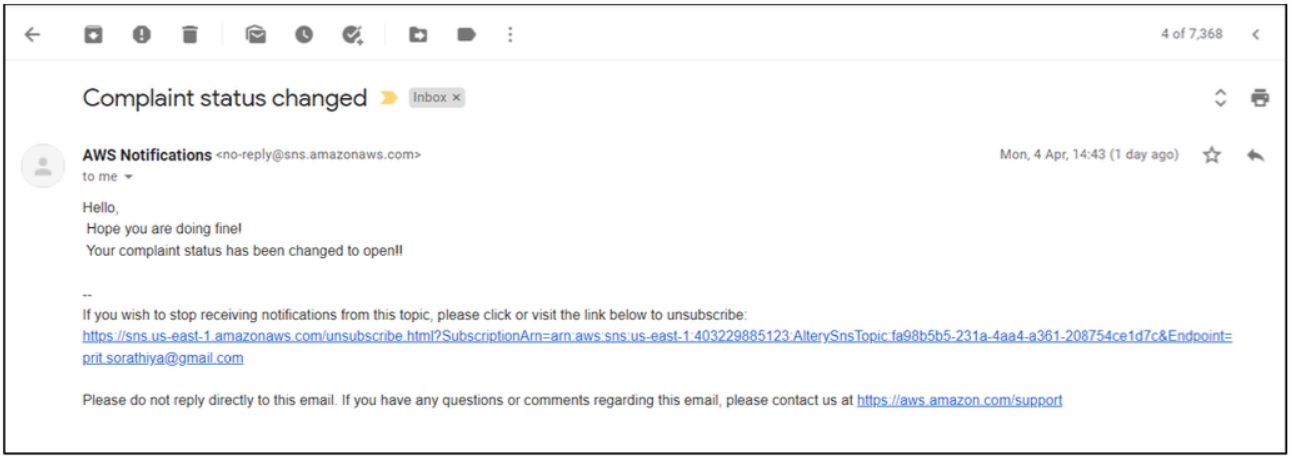

Figure 13: Email notification update to the complainer

Screenshots of the AWS Service configuration

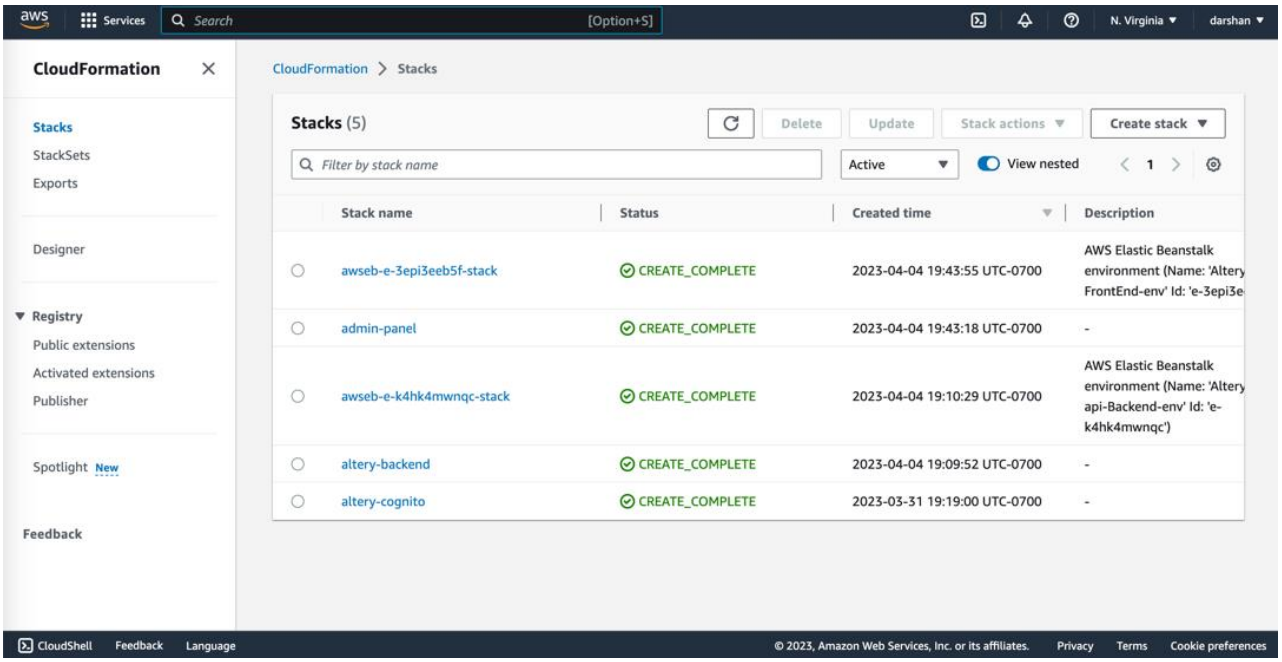

Figure 14: Screenshot of the successfully executed stack on cloud formation

Figure 10 illustrates our success in configuring all AWS services using cloud formation. Cognito, secret manager, API Gateway, Lambda as well as Elastic Beanstalk deployment for frontend and backend are all configured using YAML files (CloudFormation files).

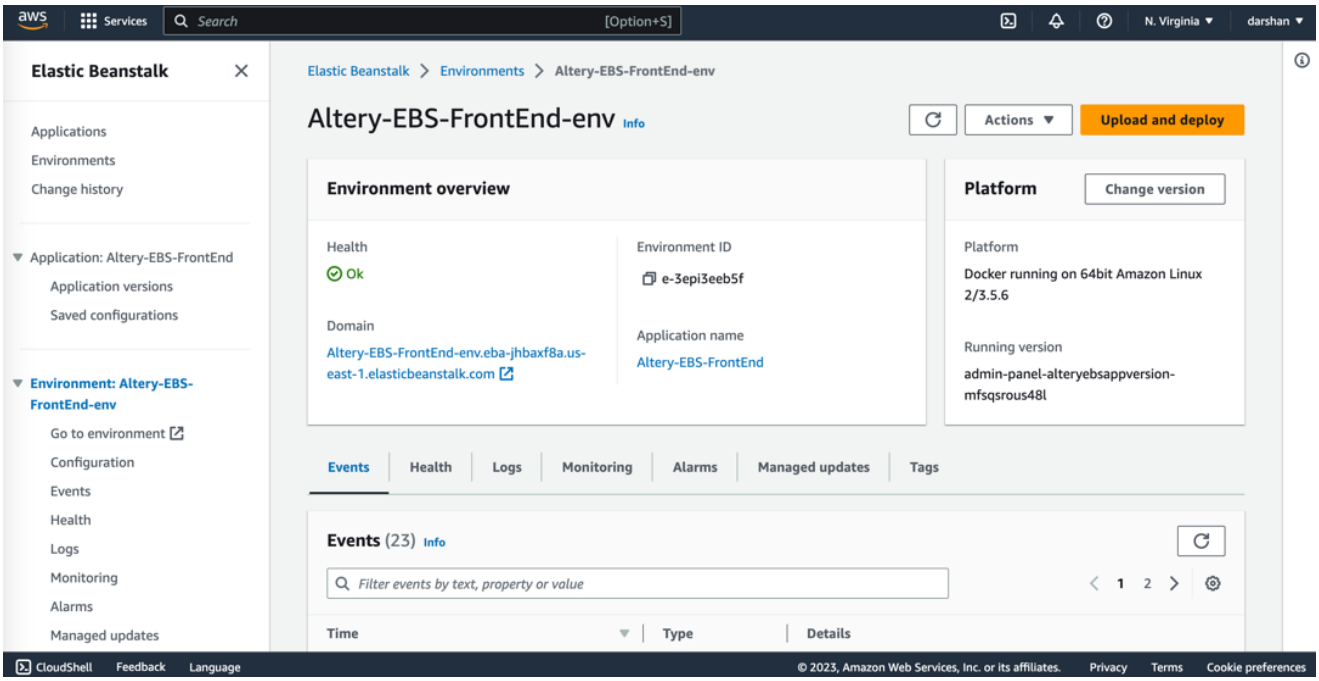

Figure 15: Front end application deployed on AWS cloud

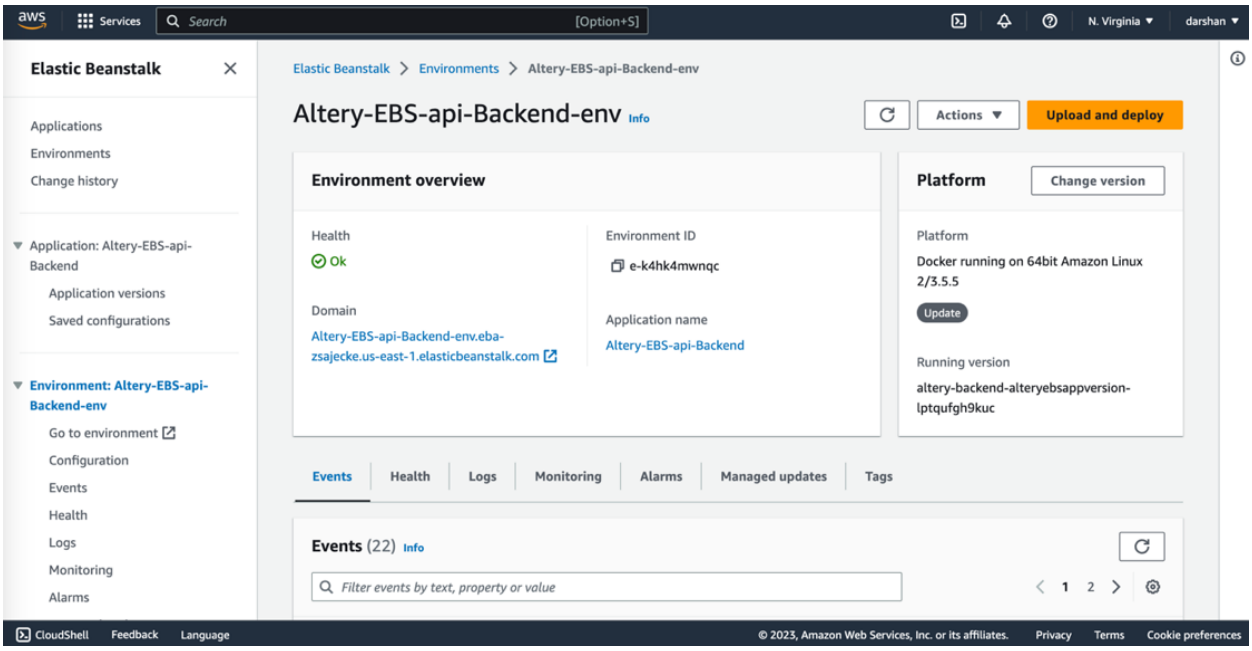

Figure 16: Backend application deployed on AWS cloud

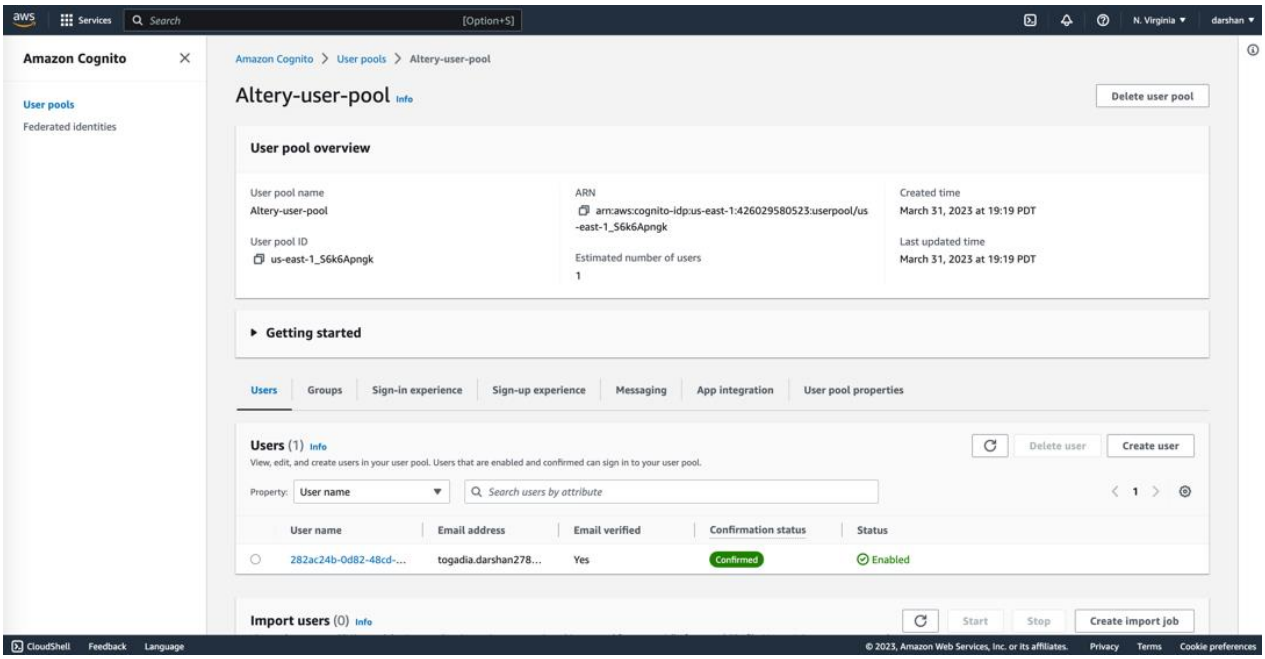

Figure 17: AWS Cognito configured user pool

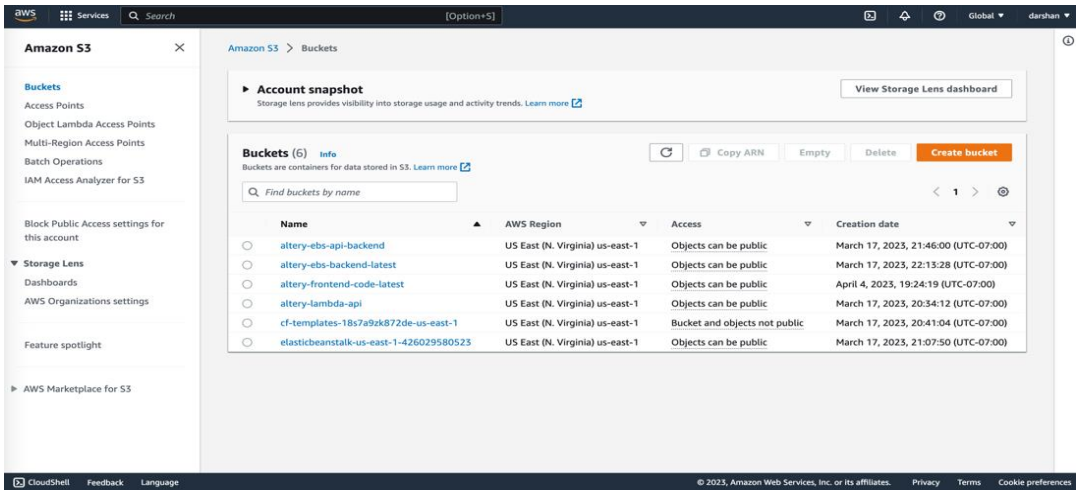

Figure 18: AWS S3 bucket configuration

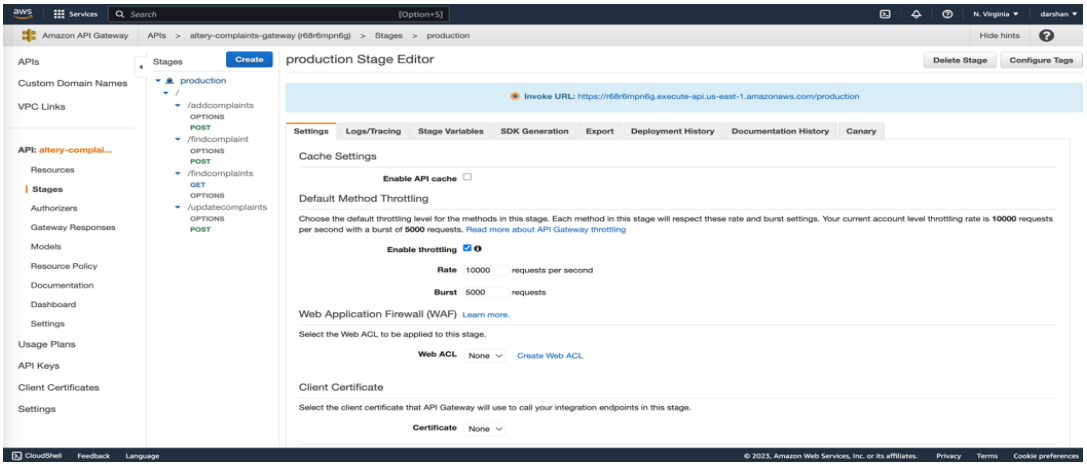

Figure 19: AWS API gateway configuration

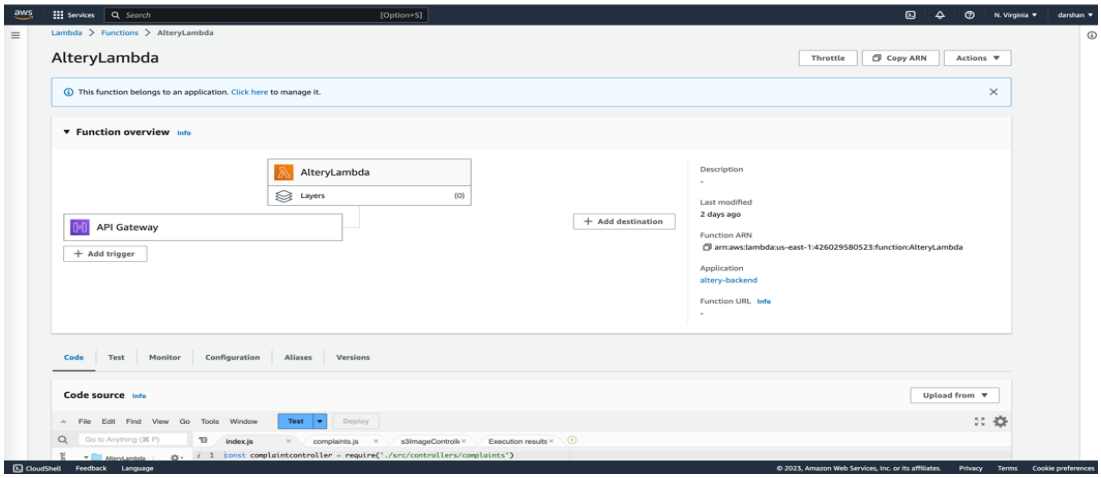

Figure 20: AWS lambda configuration with associated API gateway

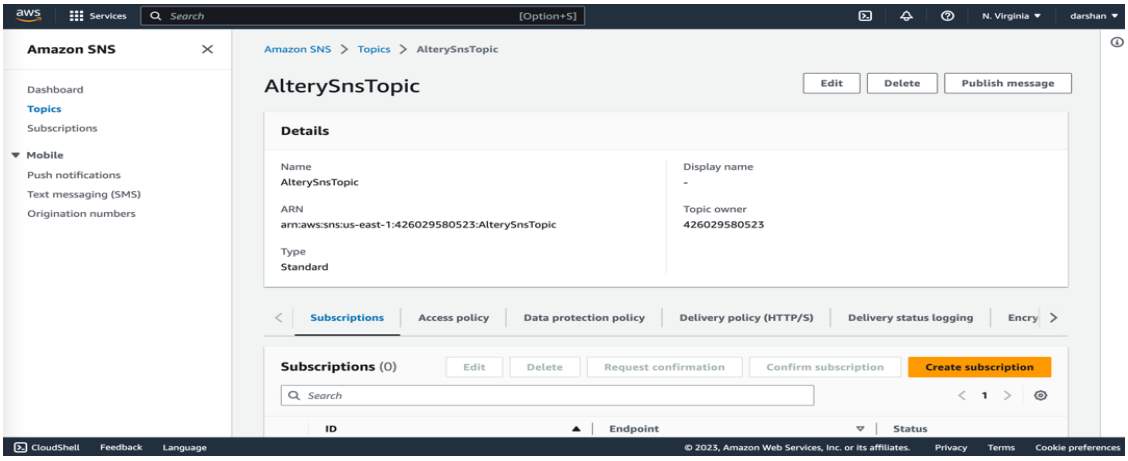

Figure 21: AWS SNS service configuration

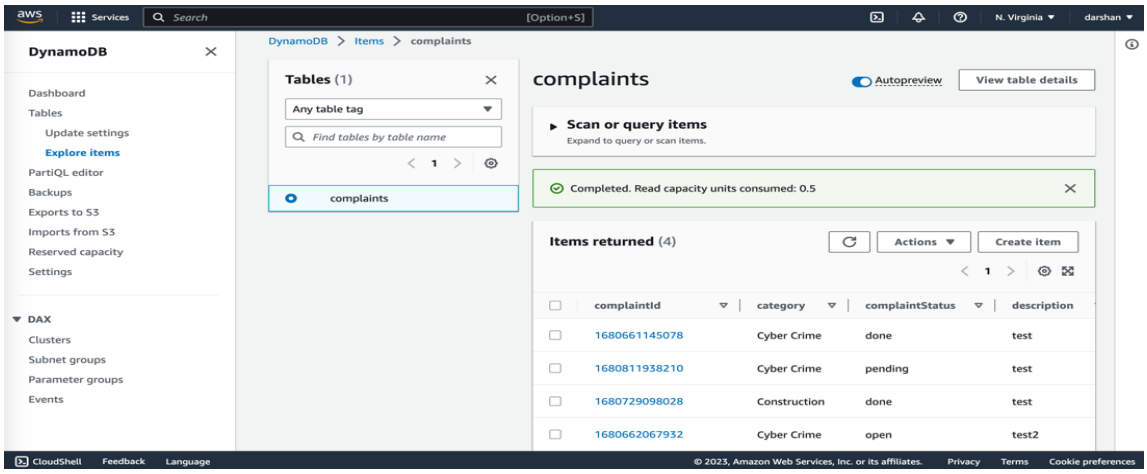

Figure 22: AWS DynamoDB configuration

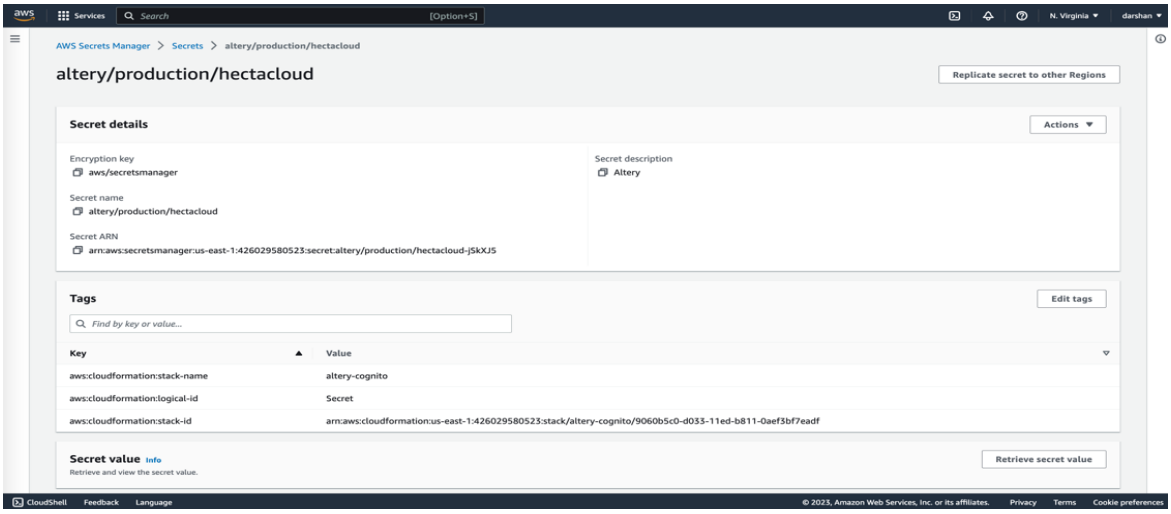

Figure 23: AWS secret manager service configuration

CHAPTER SIX

CONCLUSION

Limitations and Challenges

One of the major limitations of this project was a lack of real-world data to test the application. Since it had been created with one purpose in mind, collecting an extensive dataset of real-world complaints would not have been feasible. Instead, mock data was used which may not accurately reflect actual data and scenarios encountered in everyday use. As such, further testing and optimization may be necessary in order to guarantee that the app functions correctly and meets users' needs.

Another challenge encountered during the project was integrating third-party services. The application relied on several third-party tools, such as map and image processing services, which required additional configuration and troubleshooting. This added complexity to the development process and could have lead to increased development time and costs due to delays.

Future Work and Scope

In the future, the application can be further optimized by adding features and functionalities tailored to its users' requirements. For instance, the user feedback and reviews could be integrated into the design to improve complaint quality as well as provide valuable insights into application performance.

To categorize the complaints automatically or to rank the complaints based on severity, we can use machine learning algorithms. This will help the process to focus on priority incidents whose urgency is prime concern. Future research can be done to build more efficient serverless system. For building the effective system which can handle large data, AI and ML techniques could be used for improving accuracy and further enhancement

REFERENCES

1. "Cloud Deliver Models," [Online]. Available: https://www.exitcertified.com/blog/cloud-computing- service-delivery-models. [Accessed 13 Februrary 2022].
2. "Project Proposal," [Online]. Available: https://dal.brightspace.com/d2l/lms/dropbox/user/folder_user_view_feedback.d2l?db=140998&gr pid=215960&isprv=0&bp=0&ou=203595. [Accessed 4 April 2022].
3. "Cloud Services - Amazon Web Services," [Online]. Available: https://aws.amazon.com/. [Accessed 4 April 2022].
4. "React Native- learn once, write anywhere," [Online]. Available: https://reactnative.dev/. [Accessed 13 February 2022].
5. "Express - Node.js web application framework," [Online]. Available: https://expressjs.com/. [Accessed 13 February 2022].
6. "Elastic Beanstalk CloudFormation Template," [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/quickref-elasticbeanstalk.html.
7. "AWS Cognito CloudFormation Template," [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-cognito- userpool.html. [Accessed 4 April 2022].
8. "AWS S3 CloudFormation Template," [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket.html. [Accessed 4 April 2022].
9. "APIGateway CloudFormation Template," [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-apigateway- method.html. [Accessed 4 April 2022].

10. "AWS DynamoDB CloudFormation Template," [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-dynamodb- table.html. [Accessed 4 April 2022].
11. "Secret Manager CloudFormation Template," [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-secretsmanager- secret.html.
12. A Al-Mudimigh, M Zairi & M Al-Mashari (2001) ERP software implementation: an integrative framework, European Journal of Information Systems, 10:4, 216-226, DOI: 10.1057/palgrave.ejis.3000406