PR/5730
29.9.18

Major Project Report on

Email Listener Service

By

Kunal Agarwal

(20140007)

In partial fulfilment of requirements for the award of degree in
Bachelor of Technology in Computer Science and Engineering
(2018)

Under the Project Guidance of

Christopher Johnson, Software Development Manager, Amazon

And

Internal Reviewer

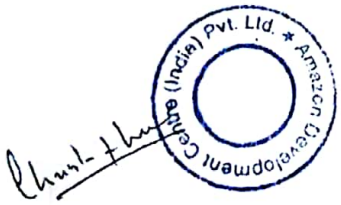
Mr. Sunil Dhimal, Assistant Professor-II, SMIT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY
(A constituent college of Sikkim Manipal University)
MAJITAR, RANGPO, EAST SIKKIM – 737136

PROJECT COMPLETION CERTIFICATE

This is to certify that the below mentioned student of Sikkim Manipal Institute of Technology has worked under my supervision and guidance from 08th January, 2018 to 30th April, 2018 and has successfully completed the project entitled "Email Listener Service" in partial fulfilment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering.

University Registration No	Name of Student(s)	Course
20140007	Kunal Agarwal	B.Tech (CSE)



Christopher Johnson

Software Development Manager (SDM)

Amazon India

PROJECT REVIEW CERTIFICATE

This is to certify that the work recorded in this project report entitled **"Email Listener Service"** has been jointly carried out by **Mr. Kunal Agarwal (Reg no. 20140007)** of Computer Science & Engineering Department of Sikkim Manipal Institute of Technology in partial fulfilment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering. This report has been duly reviewed by the undersigned and recommended for final submission for Major Project Viva Examination.

Sunil Dhimal
22/5/18

Mr. Sunil Dhimal
Assistant Professor
Department of Computer Science & Engineering
Sikkim Manipal Institute of Technology
Majitar, East Sikkim – 737136.

CERTIFICATE OF ACCEPTANCE

This is to certify that the below mentioned student of Computer Science & Engineering Department of Sikkim Manipal Institute of Technology (SMIT) has worked under the supervision of **Christopher Johnson** of **Amazon, Hyderabad** from 08 Jan 2018 to 30 April 2018 on the project entitled **"Email Listener Service".**

The project is hereby accepted by the Department of Computer Science & Engineering, SMIT in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering.

University Registration No	Name of Student(s)	Project Venue
20140007	Kunal Agarwal	Amazon, Hyderabad

22/05/18

Kalpana Sharma
Prof. & Head
Computer Science & Engineering
Sikkim Manipal Institute of Technology
Majitar, Rangpo- 737136

Dr Kalpana Sharma
Professor & HOD
Computer Science & Engineering Department
Sikkim Manipal Institute of Technology
Majhitar, Sikkim – 737136

Signature .....
Date .....

DECLARATION

I the undersigned, hereby declare that the work recorded in this project report entitled **"Email Listener Service"** in partial fulfillment for the requirements of award of B.Tech in Computer Science & Engineering from Sikkim Manipal Institute of Technology (A constituent college of Sikkim Manipal University) is a faithful and bonafide project work carried out at **Hyderabad** under the supervision and guidance of **Christopher Johnson** of **Amazon, Hyderabad**.

The results of this investigation reported in this project have so far not been reported for any other Degree / Diploma or any other Technical forum.

The assistance and help received during the course of the investigation have been duly acknowledged.

Kunal

.....

Mr. Kunal Agarwal (Reg 20140007)

ACKNOWLEDGEMENT

The internship opportunity I had with Amazon was a great chance for learning and professional development. I am also grateful for having a chance to meet so many wonderful people and professionals who led us through this internship period.

I take this opportunity to express my deepest gratitude to Christopher Johnson at Amazon, Hyderabad who in spite of being busy with his duties, took time out to hear, guide and keep me on the correct path and allowing me to carry out my project at their esteemed organization.

I wish to express my sincere gratitude to Ms. Kalpana Sharma, HoD and Professor, CSE Department, SMIT for continuously providing me support and updates about the project expectations for the entire duration of the project.

I would also take this opportunity to thank Anvesh Boddu (Mentor) for guiding me throughout the project.

Lastly, I thank my parents and my friends for their support.

Kunal

.....

Mr. Kunal Agarwal (Reg 20140007)

DOCUMENT CONTROL SHEET

1	Report No	CSE/Major Project/ External /B.Tech/61/2018
2	Title of the Report	Email Listener Service
3	Type of Report	Technical
4	Author(s)	Mr. Kunal Agarwal (20140007)
5	Organizing Unit	Amazon, Hyderabad
6	Language of the Document	English
7	Abstract	This project is aimed to develop a service that can listen to a given email domain and publish the sent attachments to a cloud storage after performing some operations.
8	Security Classification	Restricted
9	Distribution Statement	Restricted

TABLE OF CONTENTS

Title	Page No.
1. Introduction	01
1.1 General Overview	03
1.2 Problem Definition	04
1.3 Analysis of the Problem	04
1.4 Solution Strategy	06
1.5 Technical Specifications	07
1.6 Organization of the report	13
2. Design	15
2.1 High Level Design	16
2.2 Class diagram	18
2.3 Use case diagram	20
3. Detailed Test Plan	21
3.1 Unit Testing	22
3.2 Integration Testing	25
3.3 System Testing	25
3.4 Acceptance Testing	25
4. Implementation Details	26
4.1 Pseudo code	27
5. Gantt Chart	31
6. Summary and Conclusion	33
6.1 Summary of Achievements	34
6.2 Limitations	34
7. Future Scope	35
8. References	37

LIST OF FIGURES

Figure No	Figure Name	Page No.
1	Basic Workflow	06
2	High Level Design	16
3	Class Diagram	18
4	Use Case Diagram	20
5	Gantt Chart	32

LIST OF TABLES

Table No	Table Name	Page No.
1	Attachment Type Validation	22
2	Document Id Validation	23
3	User Authorization Check	23
4	Get User Details using User Name	24

LIST OF SYMBOLS AND ABBREVIATIONS

Abbreviation	Full Form
S3	Simple Storage Service
EC2	Elastic Compute Cloud
RDS	Relational Database Service
SQS	Simple Queue Service
DLQ	Dead Letter Queue
SNS	Simple Notification Service
WBR	Weekly Business Report
BIE	Business Intelligence Engineer
API	Application Programming Interface
REST	Representational State Transfer
JDK	Java Development Kit

ABSTRACT

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY
MAJITAR, RANGPO, EAST SIKKIM – 737136

ABSTRACT

Weekly Business Reports (WBRs) are generated on a timely basis which are used by Business Intelligence Engineers (BIEs) for the measurement of various metrics. The performance measurement of different metrics for each country is required for smooth functioning of the business.

The reports are generated and published to a cloud storage through different services. Since the number of metrics are very huge, while some of the reports are generated, some are directly uploaded by the business team in the cloud. It is a tedious task to manually upload a document in the cloud, thus requiring technical knowledge of database (for registering) as well as the cloud storage. It causes a huge dependency on the technical team requiring manual intervention and is error-prone.

There is a need of an Email Listener Service that can listen to a given email domain and get the documents sent to it. The service must be capable of polling the email Id at certain intervals without requiring any human effort. For each mail received, it is required to extract the mail details and publish the attachments sent to it in the cloud storage. This service helps in automating the entire process where the attachments get published to an external storage service without any manual intervention. The service must be designed in such a way that publishes only the valid emails (sent by authorized users), thereby preventing spams.

The primary goal of this project was to enable the business teams with no programming experience to get their documents published to the cloud storage just by sending an email and without requiring any technical knowledge.

Chapter 1

INTRODUCTION

1. INTRODUCTION

There is a team in Amazon, Hyderabad which is currently building tools to facilitate automation of Weekly Business Reports (WBRs) generation in a much faster rate and in an efficient way. In the current scenario, Daily and Weekly reports are generated which are used by Business Intelligence Engineers (BIEs) for performance measurement of various metrics.

The WBR Report generation project is a simple, efficient and robust reporting tool which is aimed to reduce WBR generation time to a fraction of minutes. It is bundled with powerful services which work and scale independently but collectively makes the process efficient. It is the single point of entry for the customers for measuring the performance of various metrics required for smooth functioning of the business.

The core functioning of the report generation tool is divided into 3 different services:

1. Excel Template Generator (ETG)
2. Excel Refresh Service (ERS)
3. Document Consolidator

Current system supports the following features:

- Download consolidated documents/decks.
- Download latest documents which are part of a deck.
- Download templates present in the system.
- Register a team along with a list of Business Units for it.
- Rerun Document Consolidation for a Deck.
- Register a Deck/Document

1.1 General Overview

Few of the services owned by my team which are currently being developed for building the tool are:

- **Excel Template Generation Service:** This service is used to generate the excel templates for reports dynamically reading the configuration file and using underlying concepts of the excel.
- **Excel Refresh Service:** This service is used to populate and refresh the excel sheet and send the same to document consolidator to generate a complete report.
- **Document Consolidator:** Service which consolidates all the generated excel sheets to form a WBR in PDF format.
- **Email Listener Service:** Service which continuously listens to emails in order to generate reports for the documents sent by emails.
- **Core Service:** Core service which takes care of all CRUD operations related to database which are required by all the other services of our team. All the database related operations are done through this service. Database used by my team is Aurora DB (MySQL), one of the Amazon Relational Database Service (RDS).
- **WBR Portal:** Complete portal where the team or business unit can register, modify and view all their generated reports with other functionalities as well.
- **Resource Manager:** Common utility for retrieving and publishing resources to Simple Service Storage.

The intention is to create a new service called Email Listener to read all the emails and save the email attachments (reports) for consolidation in the cloud storage.

1.2 Problem Definition

Each team has their own set of requirements for consolidating reports/documents. Although one can manually upload a document, it is a cumbersome process involving database insertion queries and then uploading it in the cloud.

Before this service, each document was required to be manually uploaded in the Simple Storage Service (S3) and also register in the database, thus requiring manual intervention.

It had the following disadvantages:

- Prone to error due to oversight
- Requires knowledge of writing SQL queries
- Requires knowledge of uploading files in S3 buckets
- Tedious task
- Manual human effort
- High latency

Considering these, there arises a need to automate the entire process where the attachments get published to external storage service (example: S3) without any manual intervention.

1.3 Analysis of the Problem

The intention here is to create a service that uploads the document into the cloud without requiring any manual intervention. It must check for the attachments sent at every specified interval. The service must listen to a given email Id and poll the queue (SQS) after every certain interval to check for the new mails. The service must keep on polling the email Id at certain intervals (depending upon the polling technique).

For each mail received, there is a need to extract the mail details and get the attachments sent to it.

This service is aimed to automate the process of uploading a newly registered document to an external storage service (example: S3) without any manual intervention. Without this service, it is a cumbersome process to manually register a document first in the portal and then upload in S3. This also does not facilitate any authorization check or logging mechanism.

The prime motive of building the service is to remove the dependency of non-technical Business team on the Technical team. Also, this will require the end-users to simply send a mail, instead of having the technical knowledge of S3 and Database, thus automating the entire process.

Requirements:

The following are some of the use-cases considered while developing the listener service:

1. Only authorized users can send report as an email attachment
2. Flexible to accept PDFs and Excel templates as an email attachment and discard all other attachments
3. All the attachments to be processed using Simple Queue Service (SQS) and stored in Simple Storage Service (S3).
4. Store email sender alias, report name and the timestamp for logging purposes
5. The correct path where the file will be stored in the cloud needs to be generated based on the report sent as an email attachment.

1.4 Solution Strategy

The overall project is designed following the OOPS principles and weekly design reviews in an iterative manner.

The development of the Listener service is broken down into 3 main modules as described below:

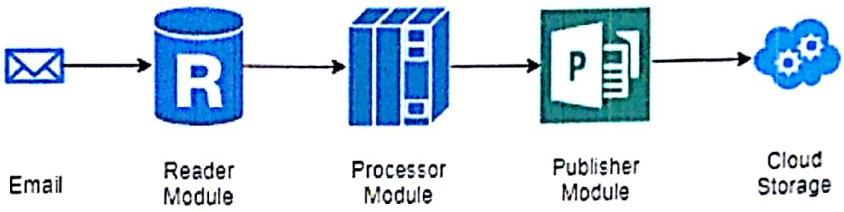


Figure 1: Basic Workflow

1. **Email Reader**: As the service kicks off, the first sole task is to initiate the Email Reader module for reading the mails. This component polls the queue to read new messages sent to the email Id. The polling method used is **Long Polling** (polling after every 20 seconds).
2. **Email Processor**: After an email is read successfully, the email is then extracted to get the sender details and the attachment. It extracts the Subject line, To Address, Sent From, Attachment and Body content from the mail and processes the email to check if the attachment is of valid type or not (i.e., PDF/Excel format). It also checks if the To Address corresponds to the format **listener+<Document Id>@email.amazon.com**. It will not call the Publisher service if any of these conditions are not satisfied.

3. **Publisher:** This component is used to publish the attachments to S3 after the Email processing is done. It makes use of Core Service Client to check if the user has access to publish the attachment or not. Request parameters include:

    - Document ID
    - User ID

Other Components:

- **Resource Manager** is used to publish the attachments to S3. The S3 path (i.e., key) where the attachment is saved in the S3 bucket is generated by the Resource Manager client based on the Document Id and file type (either PDF/Excel).
- **Core Service** is used for the following use cases (i.e., all database related operations):
    - To check if the user has access to publish the attachment or not.
    - To get user Id of a user through the email Id.
    - To log the details after attachment is published successfully.

1.5 Technical Specifications

1.5.1 Simple Storage Service (S3): Companies today need the ability to simply and securely collect, store, and analyse their data at a massive scale. Amazon S3 is object storage built to store and retrieve any amount of data from anywhere – web sites and mobile apps, corporate applications, and data from IoT sensors or devices. It is designed to deliver 99.999999999% durability, and stores data for millions of applications used by market leaders in every industry. It is the most supported cloud storage service available, with integration from the largest community of third-party solutions, systems integrator partners, and other AWS services.

Amazon S3 is intentionally built with a minimal feature set that focuses on simplicity and robustness.

Following are some of the **advantages** of Amazon S3 service:

- **Create Buckets** – Create and name a bucket that stores data. Buckets are the fundamental container in Amazon S3 for data storage.
- **Store data in Buckets** – Store an infinite amount of data in a bucket. Upload as many objects as needed into an Amazon S3 bucket. Each object can contain up to 5 TB of data. Each object is stored and retrieved using a unique developer-assigned key.
- **Download data** – Download the data or enable others to do so. Download data any time as needed or allow others to do the same.
- **Permissions** – Grant or deny access to others who want to upload or download data into one's Amazon S3 bucket.
- **Standard interfaces** – Use standards-based REST interfaces designed to work with any Internet-development toolkit.

Basic Terminologies of S3:

1. **Bucket**: A bucket is a container for objects stored in Amazon S3. Every object is contained in a bucket.
2. **Objects**: Objects are the fundamental entities stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3. The metadata is a set of name-value pairs that describe the object. An object is uniquely identified within a bucket by a key (name) and a version ID.
3. **Key**: A key is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. Because the combination of a bucket, key, and version ID uniquely identify each object, Amazon S3 can be thought of as a basic data map between "bucket + key + version" and the object itself. Every object in

Amazon S3 can be uniquely addressed through the combination of the web service endpoint, bucket name, key, and optionally, a version.

Requests are authorized using an access control list associated with each bucket and object.

Bucket names and keys are chosen so that objects are addressable using HTTP URLs:

- http://s3.amazonaws.com/bucket/key
- http://bucket.s3.amazonaws.com/key

Because objects are accessible by unmodified HTTP clients, S3 can be used to replace significant existing (static) web-hosting infrastructure. The Amazon AWS Authentication mechanism allows the bucket owner to create an authenticated URL with time-bounded validity. That is, someone can construct a URL that can be handed off to a third-party for access for a period such as the next 30 minutes, or the next 24 hours.

1.5.2 Simple Queue Service (SQS): Amazon Simple Queue Service (SQS) is a web service that gives one the access to message queues that stores the messages waiting to be processed. With Amazon SQS, one can quickly build message queuing applications that can run on any computer.

Amazon SQS offers two queue types for different application requirements:

1. **Standard Queues**: Amazon SQS offers standard as the default queue type. A standard queue lets one have a nearly-unlimited number of transactions per second. Standard queues provide best-effort ordering which ensures that messages are generally delivered in the same order as they are sent.
2. **FIFO Queues**: The FIFO queue complements the standard queue. The most important features of this queue type are FIFO (first-in-first-out) delivery and exactly-once processing: the order in which messages are sent and received is

strictly preserved and a message is delivered once and remains available until a consumer processes and deletes it; duplicates are not introduced into the queue.

Dead Letter Queues (DLQs): If one has associated a Dead Letter Queue with a source queue, messages will be moved to the Dead Letter Queue after the maximum number of delivery attempts specified have been reached. Developers can handle stuck messages (messages that have not been successfully processed by a consumer) with Dead Letter Queues. When the maximum receive count is exceeded for a message it will be moved to the Dead Letter Queue (DLQ) associated with the original queue. Developers can setup separate consumer processes for DLQs which can help analyse and understand why messages are getting stuck. DLQs must be of the same type as the source queue (standard or FIFO).

SQS Long Polling: Long polling reduces extraneous polling to help one minimize cost while receiving new messages as quickly as possible. When the queue is empty, long-poll requests wait up to 20 seconds for the next message to arrive. Long poll requests cost the same amount as regular requests.

1.5.3 Simple Notification Service (SNS): Amazon Simple Notification Service (SNS) is a flexible, fully managed messaging and mobile notifications service for coordinating the delivery of messages to subscribing endpoints and clients. With SNS one can fan-out messages to a large number of subscribers, including distributed systems and services, and mobile devices. It is easy to set up, operate, and reliably send notifications to all your endpoints – at any scale. SNS supports a variety of subscription types, allowing one to push messages directly to Amazon Simple Queue Service (SQS) queues, AWS Lambda functions, and HTTP endpoints. SNS works with SQS to provide a powerful messaging solution for building cloud applications that are fault tolerant and easy to scale.

1.5.4 Amazon Relational Database Service (RDS): Amazon Relational Database Service (or Amazon RDS) is a distributed relational database service by Amazon Web Services (AWS). It is a web service running "in the cloud" designed to simplify the setup, operation, and scaling of a relational database for use in applications. Complex administration processes like patching the database software, backing up databases and enabling point-in-time recovery are managed automatically. Scaling storage and compute resources can be performed by a single API call.

Amazon RDS was first released on 22 October 2009, supporting MySQL databases. This was followed by support for Oracle Database in June 2011, Microsoft SQL Serverin May 2012, PostgreSQL in November 2013 and MariaDB (a fork of MySQL) in October 2015.

In November 2014 AWS announced **Amazon Aurora**, a MySQL-compatible database offering enhanced high availability and performance.

Amazon Aurora is a fully managed, MySQL- and PostgreSQL-compatible, relational database engine. It combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases. It delivers up to five times the throughput of MySQL and up to three times the throughput of PostgreSQL without requiring changes to most of your existing applications.

1.5.5 Amazon Redshift: Amazon Redshift is an Internet hosting service and data warehouse product announced in November 2012 which forms part of the larger cloud-computing platform Amazon Web Services. It is built on top of technology from the massive parallel processing (MPP) data-warehouse company ParAccel, to handle large scale data setsand database migrations. Redshift differs from Amazon's other hosted database offering, Amazon RDS, in its ability to handle analytics workloads on big data data sets stored by a column-oriented DBMS principle.

Amazon Redshift is based on PostgreSQL 8.0.2. PostgreSQL 9.x.x includes features not supported in Amazon Redshift. In addition, there are important differences between Amazon Redshift SQL and PostgreSQL 8.0.2. PostgreSQL 8.0.2 was released in 2005 and PostgreSQL has seen massive development since then. Many PostgreSQL features are not supported. An initial preview beta was released on November 2012 and a full release was made available on February 15, 2013. The service can handle connections from most other applications using ODBC and JDBC connections.

1.5.6 Elastic Compute Cloud (EC2): Amazon Elastic Compute Cloud (EC2) forms a central part of Amazon's cloud-computing platform, Amazon Web Services(AWS), by allowing users to rent virtual computers on which to run their own computer applications. EC2 encourages scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image (AMI) to configure a virtual machine, which Amazon calls an "instance", containing any software desired. A user can create, launch, and terminate server-instances as needed, paying by the second for active servers – hence the term "elastic". EC2 provides users with control over the geographical location of instances that allows for latency optimization and high levels of redundancy.

In November 2010, Amazon switched its own retail website to use EC2 and AWS

1.5.7 Java (Programming Language): Java is used as a primarily language for the development of this project. Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java

applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture.

1.5.8 Mockito (Testing Framework): **Mockito** is an open source testing framework for Java released under the MIT License. The framework allows the creation of test double objects (mock objects) in automated unit tests for the purpose of test-driven development (TDD) or behaviour-driven development (BDD).

In software development there is an opportunity of ensuring that objects perform the behaviours that are expected of them. One approach is to create a test automation framework that actually exercises each of those behaviours and verifies that it performs as expected, even after it is changed. However, the requirement to create an entire testing framework is often an onerous task that requires as much effort as writing the original objects that were supposed to be tested. For that reason, developers have created mock testing frameworks. These effectively fake some external dependencies so that the object being tested has a consistent interaction with its outside dependencies. Mockito intends to streamline the delivery of these external dependencies that are not subjects of the test. A study performed in 2013 on 10,000 GitHub projects found that Mockito is the 9th most popular Java library.

1.6 Organization of the report

CHAPTER 1: Introduction

This chapter gives a brief idea regarding the report Generation tool and the services required for it. A formal description of the problem, along with a general idea of the solution strategy being used to fulfil the aim is discussed as well. Additional information regarding the technological aspect is also included.

CHAPTER 2: Design

This chapter gives a visual representation of the steps taken to develop a solution for the problem. It includes a basic architectural (high level design) diagram, use case diagram and the class diagram for the overall system.

CHAPTER 3: Detailed Test Plan

This chapter includes the testing strategies adopted to test the modules.

CHAPTER 4: Implementation Details

Gives an overview of the algorithms used to implement the different modules.

CHAPTER 5: Gantt Chart

Chart depicting proposed and achieved activities of the system with timeframes.

CHAPTER 6: Summary and Conclusion

This chapter summarizes the different modules and the results accomplished

CHAPTER 7: Future Scope

This chapter gives the brief idea of the project's future scope.

CHAPTER 8: References

The references used for the completion of the project.

Chapter 2

DESIGN

2. DESIGN

2.1 High Level Design

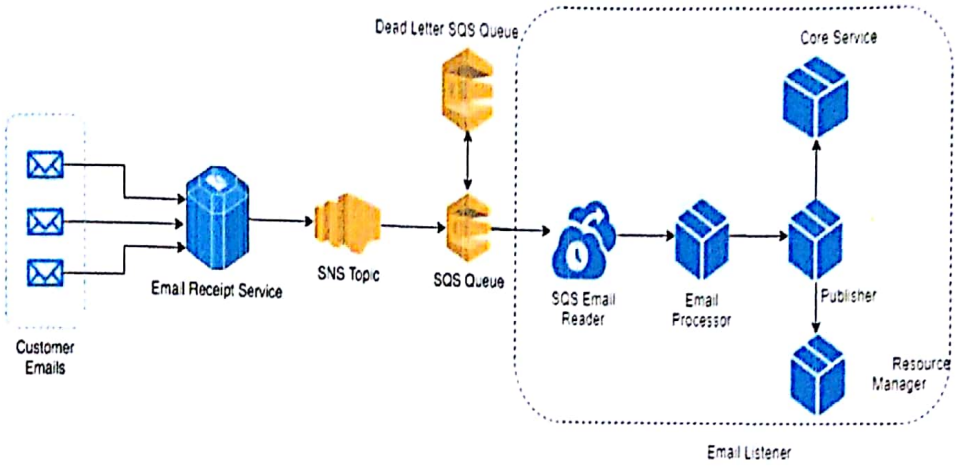


Figure 2: High Level of Email Listener Service

Detailed Description:

1. **SNS Topic:** This component is used for notification purposes. It is linked with the Email Receipt Service (with the domain: **listener@email.amazon.com**) so that whenever an email arrives, the SNS can be used as a notification service.
2. **SQS Queue:** It is a Simple Queue used to store all the Email Messages sent to the domain in a queue. It is subscribed to the SNS Topic so that all the messages sent to the email domain goes to the queue.
3. **Dead Letter SQS Queue:** It is also a simple SQS (Dead Letter Queue) to store all the unprocessed messages. In case, processing of a message fails more than 2 times, it is sent to the Dead Letter Queue.

4. **SQS Email Reader:** This component polls the SQS for the messages. After an email message arrives at the SQS, this component is used to read the email.
5. **Email Processor:** This component is mainly used to process the email content and get the sender details as well as validate the email message.
6. **Publisher:** This component is used to publish the sent attachment in the S3. It makes use of Core client for validating the user and logging the details in the Database. Resource Manager is used to publish the files to S3.
7. **Core Service:** This component is used for the authorisation purposes so that the attachments are published only if the email is sent by an authorised user. It is also used to store the logs through Core Service APIs.
8. **Resource Manager:** This component acts as a client to publish the attachment to the S3.

2.2 Class Design

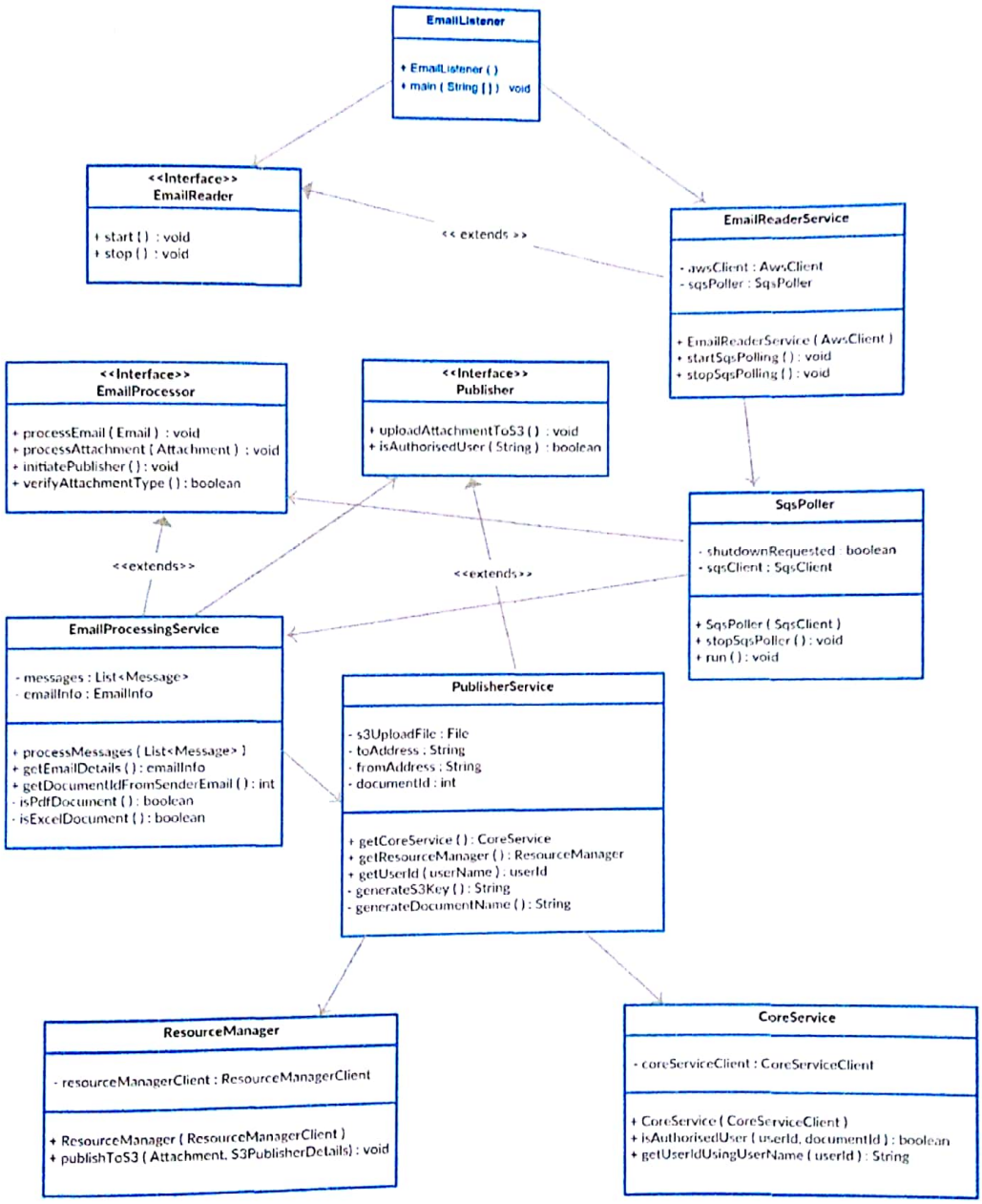


Figure 3: Class diagram

The class diagram of the Listener service is as shown above.

Detailed explanation:

1. **EmailListener**: This is the **Main** class and is responsible for initiating the Reader module.
2. **EmailReaderService**: It is the **Reader** module implementing **EmailReader** interface. Its main purpose is to poll the SQS (through **SqsPoller** class) and read the messages send to the email Id. After it reads the email message from the SQS, it then initiates the Processor module.
3. **EmailProcessingService**: It is the **Processor** Module implementing **EmailProcessor** interface. It processes and extracts the content from the received mail. It is mainly responsible for processing and validating the email sent.
4. **PublisherService**: It is the **Publisher** module implementing **Publisher** interface. Its main purpose is to publish the attachments in S3. It makes use of **CoreService** for validating user authorization and **ResourceManager** for publishing the document into the S3.

2.3 Use-case diagram

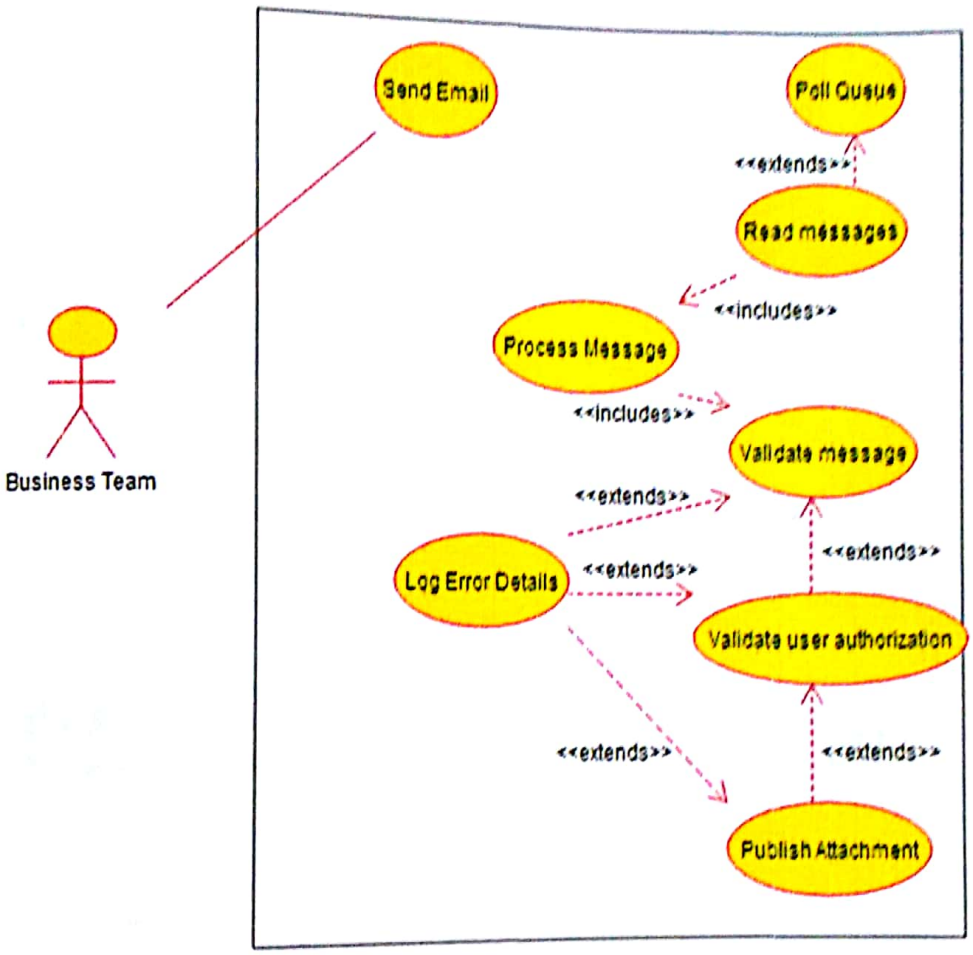


Figure 4: Use case diagram

Chapter 3

DETAILED TEST PLAN

3. DETAILED TEST PLAN

3.1 Unit Testing

All the core modules have been tested using Junit and Mockito framework.

Since, unit testing is for testing each of the individual basic components, therefore for testing purpose, the Email domain created is: **listener-integ@email.amazon.com**

- **Testcase**: Attachment Type

Sl No.	Test Case	Response
1.	Attachment Name: "Report.doc"	Exception thrown and logged in the database.
2.	Attachment Name: "Report.xlsx"	Publisher Module is initiated for publishing
3.	Attachment Name: "Report.pdf"	Publisher Module is initiated for publishing
4.	Attachment: null	Exception thrown and logged in the database.

Table 1: Attachment Type Validation

- **Testcase**: Document Id validation

Sl No.	Test Case	Response
1.	Sent To Email Field: "listener-integ@email.amazon.com" with no document Id	Exception thrown and logged in the database.
2.	Sent To Email Field: "listener-integ+1e3f3@email.amazon.com" with invalid document Id	Exception thrown and logged in the database.
3.	Sent To Email Field: "listener-integ+133@email.amazon.com" with valid document Id	Continue and then validate for attachment type
4.	Sent To Email Field: null	Exception thrown and logged in the database.

Table 2: Document Id Validation

- **Testcase**: User authorization

Sl No.	Test Case	Response
1.	User sending the attachment does not have access permission	Exception thrown and logged in the database.

2.	User sending the attachment does not exist in the database	Exception thrown and logged in the database.
3.	User sending the attachment has the access permission	Attachment is published in the S3

Table 3: User authorization check

- **Testcase**: Get User Details using User Name

Sl No.	Test Case	Response
1.	Invalid Username	Exception thrown and logged in the database.
2.	Valid Username	User Details
3.	Null Username	Exception thrown and logged in the database.

Table 4: Get User Details using User Name

3.2 Integration Testing

Integration testing focuses on unit tested modules and build the program structure that is dictated by the design phase. Each module was integrated successfully. After the integration of all the modules of the listener service, few more unit test cases were required to be added considering the edge cases (i.e., raising an exception if the message is unprocessed). Few of the files were modified such that they don't overlap each other and the system was tested successfully.

3.3 System Testing

System testing tests the integration of each module in the system. It also tests to find discrepancies between the system and its original objective, current specification and system documentation. The primary concern is the compatibility of individual modules. Entire system is working properly or not will be tested here, and giving output or not are tested here. These verifications and validations are done by giving input values to the system and by comparing with expected output. The entire system was hence, tested successfully on an Amazon EC2 host simulating the entire workflow.

3.4 Acceptance Testing

This testing was done by the project supervisor checking the entire functionality.

Chapter 4

IMPLEMENTATION DETAILS

4. IMPLEMENTATION DETAILS

4.1 Pseudo code

4.1.1 Document Id Validation:

```
Begin
    emailId = getEmailId()
    //perform regex pattern matching on emailId as
    "listener+<number>@email.amazon.com
    matchFound = match(emailId, requiredEmailFormat)
    If (matchFound) then
        //check for Attachment Type
    Else
    {
        //log details and throw exception
        throw exception after logging the sender details in DB
    }
End
```

4.1.2 Attachment Type Validation:

```
Begin
    attachment = getAttachment()
    If (attachment is of type PDF or Excel) then
```

```
            //initiate Publisher Module
        Else
        {
            //log details and throw exception
            throw exception after logging the sender details in DB
        }
    End
```

4.1.3 User Authorisation:

```
    Begin
        userName = getSenderName()
        userId = getUserId(userName)
        If (userId != -1) then
        {
            //check for access permissions
            //publish Attachment in S3
        }
        Else
        {
            //log details and throw exception
            throw exception after logging the sender details in DB
        }
```

```
End
```

4.1.4 Reading message from SQS:

```
Begin
        while(true)
        {
                messages = readMessage()
                If (messages.count > 0) then
                        //initiate Processor Module
        }
End
```

4.1.5 Publishing attachment to S3:

```
Begin
        resourceManager = getResourceManagerClient
        isPublished = resourceManager.publish(attachment)
        If (isPublished == true) then
                //log details
        else
        {
                //log details and throw exception
                throw exception after logging the sender details in DB
```

}

End

Chapter 5

GANTT CHART

5. GANTT CHART

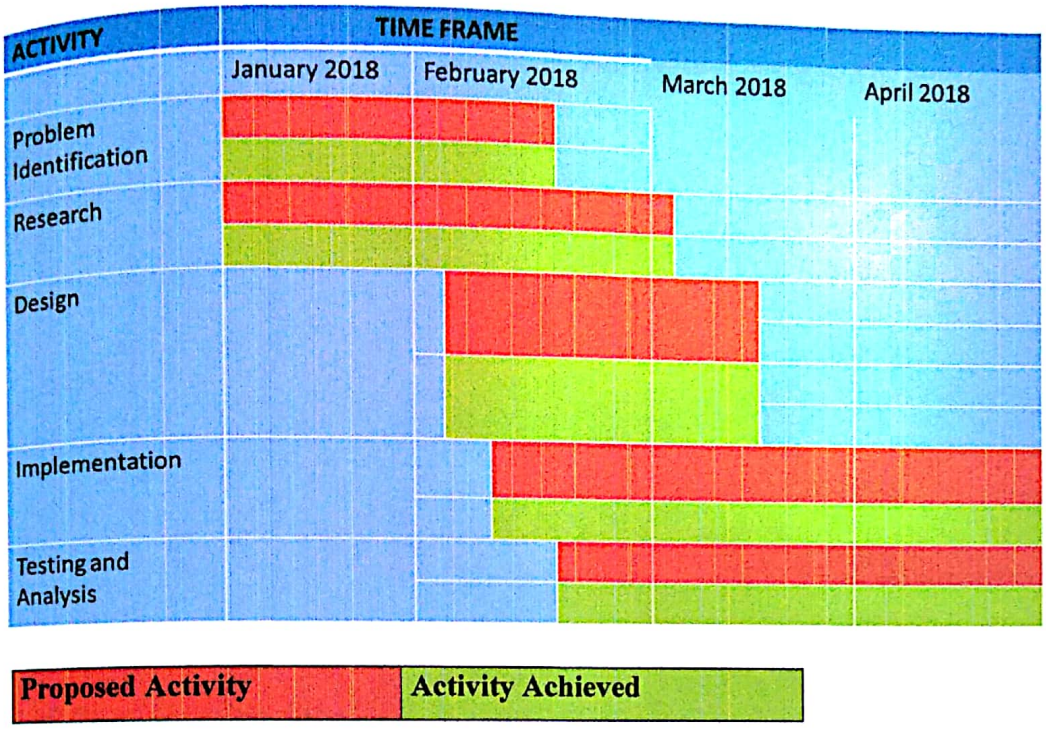


Figure 5: Gantt Chart

Chapter 6

SUMMARY AND CONCLUSION

6. SUMMARY AND CONCLUSION

6.1 Summary of Achievements

The Project was successfully completed by end of the internship.

The core functionalities of the Listener service are developed and tested successfully. All the modules such as Reader, Processor and Publisher have been validated considering the different scenarios and the Listener service is currently in production in an Elastic Compute Cloud (EC2) instance.

Users (such as Business Intelligence Engineers, BIEs) can send their reports as emails to the registered email domain for publishing the report to the cloud storage i.e., S3.

6.2 Limitations

- It publishes only PDF and Excel attachments. All other type of attachments is discarded.
- In case any exception occurs, the sender is not notified of the same. For example: if the attachment type is not of PDF of Excel, an exception is thrown and sender does not get to know whether the publishing was successful or not.
- If a sender sends more than one attachment (i.e., if the mail contains multiple attachments), the service fails to upload the correct attachment. It does not have the logic to handle multiple attachments case.

Chapter 7

FUTURE SCOPE

7. FUTURE SCOPE

Although the project is completed successfully and deployed in production, but still as a future scope of the project, the following objective can be implemented:

- **To Check the feasibility of moving the entire service to AWS**:
  It is a Migration to Native AWS - It will be a good option to move entire service to AWS. Here, the entire service is aimed to be moved to the Native AWS (Lambda), thus requiring no host(s) and just a Lambda service for computing.

  Reasons for moving the entire service to AWS:
  AWS Lambda lets one run code without provisioning or managing servers. We pay only for the compute time we consume - there is no charge when the code is not running.
  With Lambda, we can run code for virtually any type of application or backend service - all with zero administration. Just upload the code and Lambda takes care of everything required to run and scale the code with high availability. We can set up our code to automatically trigger from other AWS services or call it directly from any web or mobile app.

- **Notifying the sender**:
  There needs to be a mechanism where the sender is notified if the attachment is published or even if an error is thrown.

- **Handling multiple attachments case**:
  The service can be modified such that it handles the multiple attachments case and publishes all of them into the cloud storage.

Chapter 8

REFERENCES

8. REFERENCES

[1] S3 User Guide, https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html, [online], accessed on 20th Jan, 2018.

[2] S3 Developer SDK for Java, https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-s3.html, [online], accessed on 24th Jan, 2018.

[3] SQS Developer Guide, https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html, [online], accessed on 05th Feb, 2018.

[4] SQS SDK for Java, https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-sqs.html, [online], accessed on 21st Feb, 2018.

[5] Dead Letter SQS, https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html, [online], accessed on 28th Feb, 2018.

[6] SQS Long Polling, https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-long-polling.html, [online], accessed on 06th March, 2018.

[7] SNS, https://docs.aws.amazon.com/sns/latest/dg/GettingStarted.html, [online], accessed on 19th March, 2018.

[8] Amazon EC2, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html, [online], accessed on 29th March, 2018.

[9] EC2 Autoscaling, https://aws.amazon.com/ec2/autoscaling/?nc2=h_m1, [online], accessed on 04th April, 2018.

[10] Amazon Lambda, https://aws.amazon.com/lambda/?nc2=h_m1, [online], accessed on 04th April 2018.

[11] Redshift database, https://docs.aws.amazon.com/redshift/latest/mgmt/welcome.html, [online], accessed on 15th Feb, 2018..

[12] Amazon Aurora Database, https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Overview.html, [online], accessed on 15th Feb, 2018.

[13] Amazon RDS Documentation, https://aws.amazon.com/rds/, [online], accessed on 20th Feb, 2018.

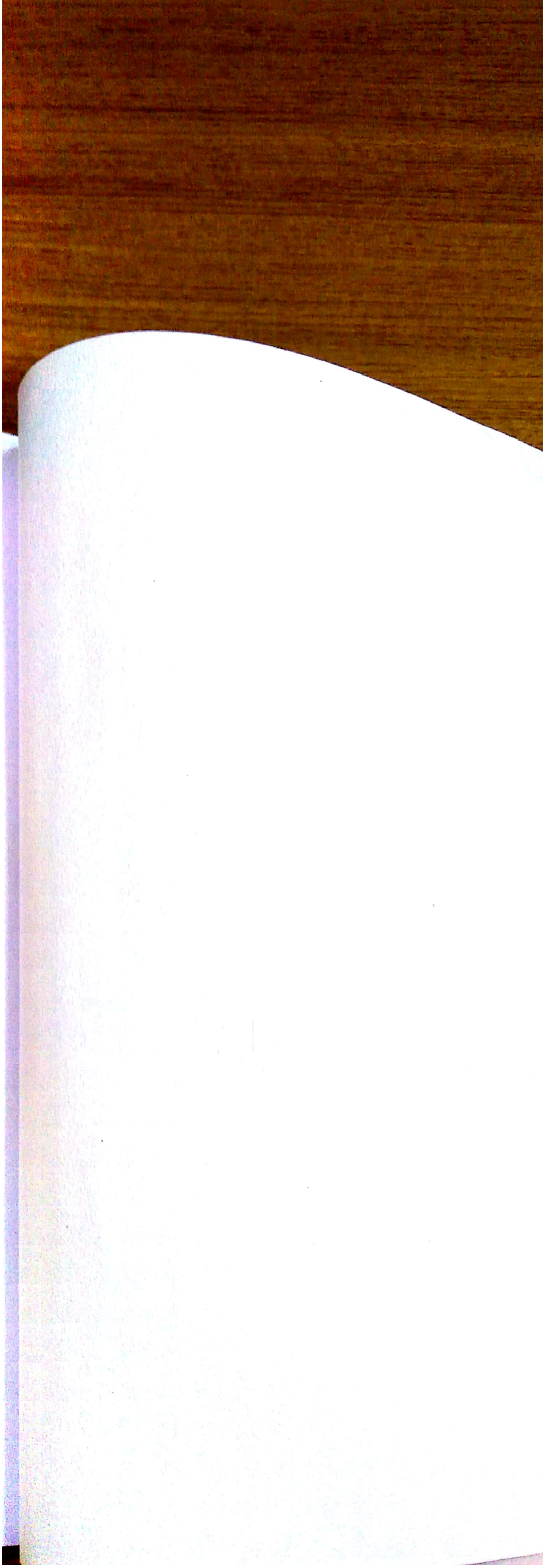
[14] AWS SDK for Java, https://github.com/aws/aws-sdk-java, [online], accessed on 28th Feb, 2018.

[15] Junit Framework, https://www.tutorialspoint.com/junit/junit_test_framework.htm, [online], accessed on 30th March, 2018.

[16] Mockito, http://site.mockito.org/, [online], accessed on 04th April, 2018.

[17] System Design, https://www.tutorialspoint.com/system_analysis_and_design/system_analysis_and_design_overview.htm, [online], accessed on 30th March, 2018.

[18] Rajib Mall, Fundamentals of Software Engineering. Phi Learning Private Limited, India (2003).



Major Project Report on

Email Listener Service

By

Kunal Agarwal

(20140007)

In partial fulfilment of requirements for the award of degree in
Bachelor of Technology in Computer Science and Engineering
(2018)

Under the Project Guidance of

Christopher Johnson, Software Development Manager, Amazon

And

Internal Reviewer

Mr. Sunil Dhimal, Assistant Professor-II, SMIT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY

(A constituent college of Sikkim Manipal University)

MAJITAR, RANGPO, EAST SIKKIM – 737136