SDE Intern - Amazon

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

DEV PRATAP TYAGI (191306)

UNDER THE SUPERVISION OF

Dr. Amit Kumar

Assistant Professor, Deptt. Of CSE & IT

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

DECLARATION

I hereby declare that the work presented in this report entitled **"SDE Intern - Amazon"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of **Dr. Amit Kumar** (Assistant Professor(SG), Department of CSE Jaypee University of Information Technology). The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Dev Pratap Tyagi
191306

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)
Dr. Amit Kumar
Assistant Professor (SG)
Department of CSE Jaypee University of Information Technology

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ……………………….

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: __________________________ __**Department:** ________________ **Enrolment No** _________

Contact No. ____________________________**E-mail.** ____________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _______________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ……………….(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

………

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to Almighty God for His divine blessing that made it possible to complete the project work successfully. I am grateful and wish my profound indebtedness to Supervisor **Dr. Amit Kumar**, Assistant Professor(SG), Department of CSE Jaypee University of Information Technology, Wakhnaghat. His deep knowledge & keen interest in the field of "Cloud Computing and Technologies" helped us to carry out this Project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this Project. I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking. Finally, I must acknowledge with due respect the constant support and patience of my parents.

Dev Pratap Tyagi
191306

Table of Figures

Table of Contents

Abstract

This work aims to re-architecture the legacy Invoice Ingestion Platform into multiple microservices, including the Matcher project. Matcher is a part of the Data Ingestion platform in the Invoicing Team at Amazon’s Transportation Financial Systems org. We divide our problem into sub-goals, design, implementation, and development, and create some tasks in these sub-goals to achieve the solution to the problem. For each sub-goal, we conduct a series of experiments and test runs to ensure the maximum accuracy of the application. We explore different and latest technologies to develop the features of the project to ensure the best quality and sustainability of the software.

Chapter 01: INTRODUCTION

1.1 Introduction

In transportation, invoices are typically used to bill customers for the transportation services provided. The invoice serves as a legal agreement between the transportation provider and the customer, outlining the terms of the transaction and the payment due. The invoice may also include additional charges, such as fuel surcharges, insurance, or other fees.

Transportation companies may use electronic invoicing systems to automate the process of generating and sending invoices to customers. Electronic invoicing can help to reduce errors and streamline the invoicing process, as well as provide faster and more accurate billing to customers. Additionally, electronic invoicing can help to improve the tracking and management of transportation transactions, which can be useful for financial reporting and analysis purposes.

Invoice ingestion is the process of automatically extracting data from an invoice and entering it into a digital system for further processing. This process typically involves using optical character recognition (OCR) technology to recognize and extract data from scanned or digital invoices, such as invoice number, date, supplier name, and itemized line items. The extracted data can then be used for various purposes such as accounts payable processing, inventory management, financial reporting, and analysis.

Invoice validation with manifests refers to the process of cross-referencing invoice data with data from shipment manifests to verify the accuracy and completeness of invoiced items. A shipment manifest is a document that lists the items included in a shipment, typically including details such as item descriptions, quantities, weights, and other relevant information. This can be done manually by comparing the invoice data with the manifest data line by line, or it can be automated using software or tools that can automatically match and validate the data from the invoice against the corresponding data in the manifest. This process helps to improve accuracy and efficiency, reduce errors, and ensure that invoices are properly validated before payment is made.

1.1.1 Outline of the report

This report is organised in four main parts:

Chapter 1 describes the outline of the complete internship project and how it is carried out.

Chapter 2 describes the study of the existing systems that do the job, and why they needs to be rearchitected into the new system.

Chapter 3 presents the analysis, design, and implementation of this project. This chapter summarises the design specifications and technical requirements of the system being proposed for this project.

Chapter 4 presents the implementation methods of the various submodules of this project.

Chapter 5 discusses the results of the system designed in this project and its limitations in terms of accuracy.

Chapter 6 is the conclusion of the project which also presents the future work and direction for the project.

1.2 Problem Statement

1.2.1 Overview

Matcher, as the name suggests, is a piece of software that does some matching. In the context of transportation invoices, this service matches the carrier invoices being ingested by other services into Amazon systems with the shipping manifests.

1.2.2 Purpose

Loosely coupled services refer to a software architecture pattern in which components or services are designed to be independent and have minimal dependencies on each other. In this pattern, each service is designed to perform a specific function or task and can operate independently of other services. With this motivation of rearchitecting the legacy service of Invoice Ingestion, the matching component is being separated out as a new component Matcher so that the new service can be used all across Amazon as this step is common for any invoicing process.

1.2.3 Features

This project has many features, but this document focuses on the features that the author has worked and contributed on to the project:

1. Dynamic Configurations: Dynamic configurations refer to a software design pattern in which application settings or configurations are designed to be flexible and adaptable to changing conditions or requirements. In this pattern, configuration parameters are stored outside of the application code, often in a database or configuration file. They can be modified at runtime without requiring a restart of the application.
2. Data Warehousing: The data warehouse is designed to support complex queries and data analysis, allowing business users to identify patterns, trends, and insights that are not readily apparent in individual data sources. It typically includes historical data that can be used to track trends over time and make more informed decisions about future operations. This project also includes data warehousing where we store and publish the data enriched by the new service. This ensures the backward compatibility of the existing flow and the required data is available for analysis of the invoicing systems.

1.3 Existing System Brief Motivation

The existing system for an organisation may involve a manual process for matching invoices with shipping manifests or related documents. This process can be time-consuming, error-prone, and may lead to discrepancies in payments or delays in processing. By implementing a manifest matching service, we streamline this process and reduce the risk of errors or discrepancies. The manifest matching service can automatically compare invoices with shipping manifests or related documents to ensure that the items listed on the invoice match the items that were shipped. This can help to prevent overpayments, underpayments, or other financial irregularities, and can reduce the workload of accounts payable personnel. In addition, a manifest matching service can help to improve operational efficiency and reduce costs associated with manual invoice processing. By automating the matching process, the organization can process invoices more quickly and accurately, which can lead to improved cash flow and better financial management.

1.4 Project Management Plan

Project planning is crucial in assisting stakeholders, sponsors, teams, and the project manager as they go through subsequent project phases. To establish desired objectives, lower risks, prevent missed deadlines, and finally deliver the agreed-upon good, service, or outcome, planning is necessary. Project performance will almost surely suffer without adequate planning. A bigger project must be broken down into smaller jobs, a project team must be put together, and a timeline must be set for when the work is to be finished. In this phase, you establish more manageable objectives within the overall project, making sure that each is feasible within the allotted time. A few tasks were delegated from this project, each having its own execution schedule, deadline, and plan.

1.4.1 Roadmap for the Internship

Figure 1.1 shows the timeline for the internship to date. Since some tasks are confidential, the task details are not mentioned in the timeline. The internship started with
a 1-week training and a ramp-up task.

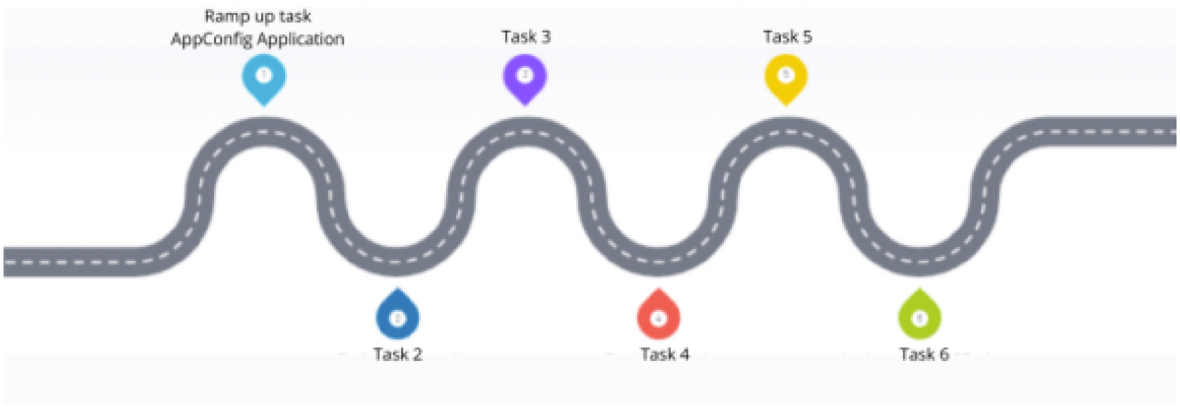

Figure 1.1: Roadmap to the Project

1.4.2 Gantt Chart

Gantt charts are useful for planning and scheduling projects. They help you assess how long a project should take, determine the resources needed, and plan the order in which the tasks are to be completed.

Below is the Gantt chart, which shows the management plan for this project.

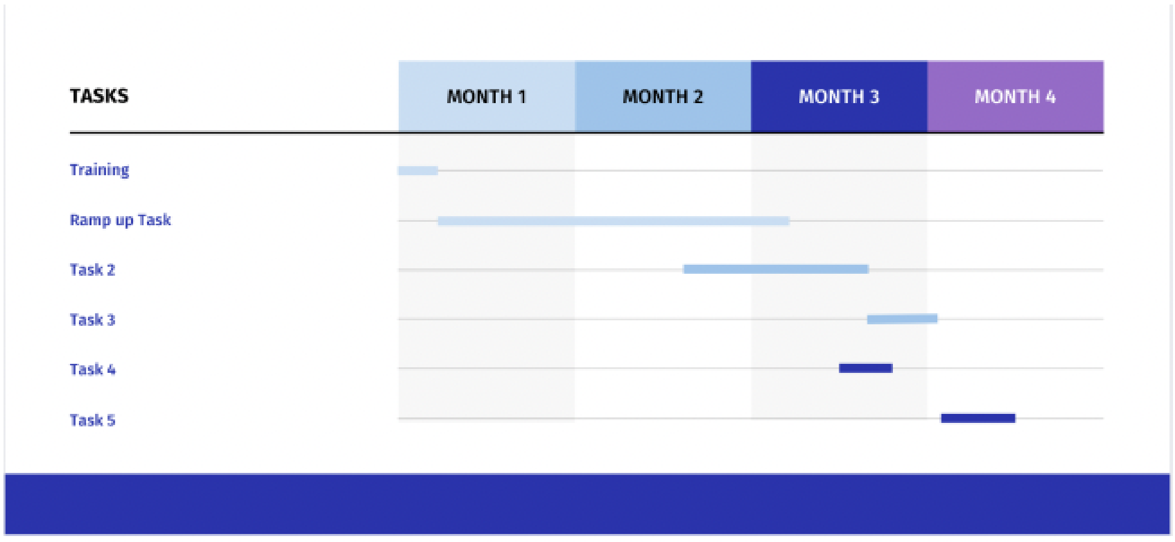

Figure 1.2: Gantt chart displaying the timelines for the different phases of the project

As can be seen from Figure 1.2, the whole project has been slotted into various phases, which are Research, Problem Understanding, Algorithm Design, Implementation, Testing, etc. All the tasks within these phases have been assigned some timeline in which those tasks are supposed to be completed. Being an industrial project, more time is given to the review process be it code review, HLD or LLD reviews, etc. to ensure the accuracy and correctness of each task being completed.

The major tasks are discussed in the further chapters of this document.

1.5 My Contribution

This is a team and internship project, where the contribution of the author is to complete the tasks with accuracy in the given timeline and to demonstrate the Amazon Leadership Principles that are being discussed in Appendix A.

1.6 Summary

This chapter is the introduction to the project, the purpose of the project, and its features.
This chapter also defines the layout of the report.
Project Management Plan and its importance along with the phases of the project are also discussed in this chapter.

Chapter 02: RELATED WORK

Efficient and accurate invoicing is critical for transportation companies to ensure timely payment and maintain healthy cash flow. Invoicing errors or delays can lead to payment delays, revenue loss, and strained customer relationships. To streamline the invoicing process, many transportation companies leverage advanced technologies, such as automated billing systems, electronic payment processing, and real-time reporting and analytics. Invoicing in transportation also involves regulatory compliance and adherence to industry standards. For example, in the United States, transportation companies must comply with the Federal Motor Carrier Safety Administration (FMCSA) regulations, which require accurate and timely invoicing and documentation of transportation services provided. Similarly, international transportation companies must comply with customs regulations and document requirements specific to each country they operate in.

2.1 Invoice Ingestion

Invoice ingestion refers to the process of extracting relevant data from invoices and integrating it into a system for further processing or analysis. This process is typically done using automated tools, such as optical character recognition (OCR) software, that can extract key information from invoices, such as invoice number, vendor name, invoice date, line item details, and total amount.

Invoice ingestion typically involves several steps, which may include:

1. Scanning or uploading invoices: Invoices can be scanned or uploaded into a system that is capable of processing invoices. This can be done using a scanner, a multifunction printer, or by uploading digital invoices from a computer or other electronic device.

2. Optical character recognition (OCR): OCR software is used to automatically recognize and extract relevant data from the invoices. OCR technology scans the text on the invoices and converts it into machine-readable text that can be processed by software.

3. Data extraction: Once the text has been converted into a machine-readable format, the relevant data, such as invoice number, vendor name, invoice date, line item details, and total amount, are extracted using data extraction techniques. This can involve using pre-defined templates or rules or employing machine learning algorithms to identify and extract the relevant information.

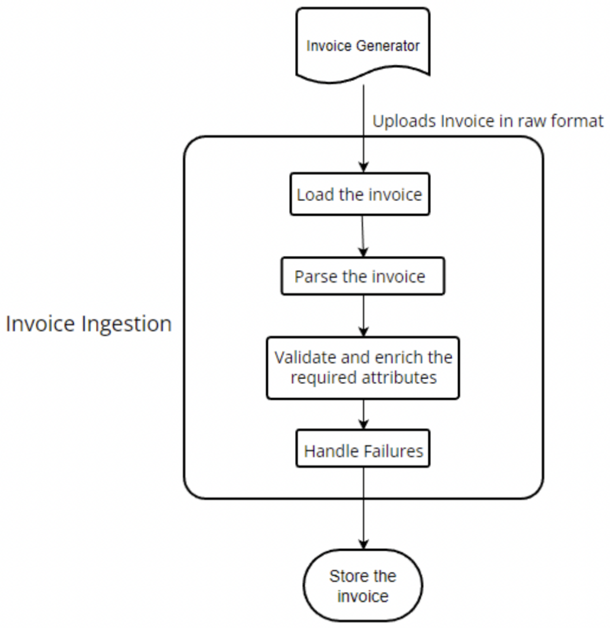

Figure 2.1: Invoice Ingestion Process

4. Data validation and verification: Extracted data may need to be validated and verified to ensure accuracy and consistency. This can involve checking extracted data against known data sources, such as vendor databases or previous invoices, and validating the data format and structure to ensure it is in the expected format.

5. Data integration: The extracted and validated data can then be integrated into the appropriate system or software for further processing, such as an accounting system or an enterprise resource planning (ERP) system. This integration can be done using APIs, data connectors, or other integration methods to transfer the data from the invoice ingestion system to the target system.

6. Exception handling: In some cases, invoices may contain exceptions or discrepancies that require manual intervention or review. These exceptions, such as

missing or ambiguous data, may need to be flagged for further investigation or resolution.

Invoice ingestion automation can streamline the invoice processing workflow, reduce manual effort, and minimize errors associated with manual data entry. It is commonly used in accounts payable (AP) departments of organizations to process large volumes of invoices efficiently and accurately.

In the context of transportation, an invoice can be a document that outlines the charges associated with the transportation of goods or services provided by a transportation service provider. It serves as a formal request for payment from the transportation provider to the customer, typically the shipper or consignee, for the services rendered.

An invoice in transportation typically includes information such as:

1. Shipment details: This includes the description of the goods being transported, the quantity or weight of the goods, and any other relevant details such as the origin and destination of the shipment.
2. Charges and rates: This includes the transportation charges, such as freight charges, handling fees, surcharges, and any other applicable charges based on the agreed upon rates or tariffs.
3. Additional services: This includes any additional services provided by the transportation provider, such as insurance, storage, or customs clearance, if applicable.
4. Terms of payment: This includes the agreed-upon terms of payment, such as due date, payment method, and any applicable discounts or penalties for late payment.
5. Contact information: This includes the contact information of both the transportation provider and the customer, including their names, addresses, phone numbers, and email addresses for communication and payment purposes.

Invoices in transportation are typically used for billing and payment purposes, and they play a crucial role in the financial management of transportation services. They provide a formal record of the services provided, the charges associated with those services, and the terms of payment, which helps ensure accurate billing and timely payment for transportation services rendered.

2.2 Manifest Matching

Invoice manifests refer to documents or records that provide a detailed listing of items or goods included in an invoice. An invoice manifest typically contains information such as item descriptions, quantities, prices, and other relevant details related to the goods or services being billed.

Invoice manifests can be used for various purposes in business, particularly in the transportation and logistics industry. For example, in the context of freight transportation, an invoice manifest may be used to list the individual items or packages that are being shipped, along with their corresponding details such as weights, dimensions, and shipping instructions. The invoice manifest serves as documentation of the goods being transported and provides the basis for generating invoices for transportation services.

Invoice manifests are typically generated by the shipper or logistics provider and provided to the customer as part of the billing process. The customer can then use the invoice manifest to verify the accuracy and completeness of the items listed on the invoice before making payment. Invoice manifests are also used for record-keeping purposes, as they provide clear documentation of the goods or services provided, which can be useful for financial, regulatory, or audit purposes.

In some cases, invoice manifests may be generated electronically using specialized software or systems that automate the process of creating and managing invoice manifest data. This can help improve the accuracy, efficiency, and speed of the invoicing process, and can also facilitate electronic data interchange (EDI) with trading partners for seamless invoicing and payment transactions.

In many businesses, the accounts payable process involves matching invoices to shipping manifests to ensure that the goods or services listed on the invoice have been received and are accurate. This process can be time-consuming and error-prone, particularly when dealing with large volumes of invoices and manifests.

Manifest matching is an important process in logistics and supply chain management that involves matching the items listed on a shipping manifest with the actual items that are being transported. This process ensures that the correct items are delivered to the correct location and that any discrepancies are identified and addressed promptly.

There are various roles and responsibilities related to manifest matching, depending on the specific context and industry. In a warehouse or distribution center, for example, workers may be responsible for physically checking the items against the manifest, verifying that the items are in good condition, and updating the manifest as needed.

In the context of software development or automation, manifest matching may involve the use of technology to compare the items listed on the manifest with data from sensors or other monitoring devices. This technology can help to automate the manifest matching process, reducing the risk of errors and improving efficiency.

In addition to ensuring the accuracy of shipments, manifest matching can also help to improve inventory management, identify trends or patterns in shipping data, and provide insights into the performance of logistics and supply chain operations.

Some common tasks related to manifest matching include:

- Receiving and reviewing shipping manifests
- Physically checking items against the manifest
- Using technology to automate the manifest matching process
- Resolving any discrepancies or issues with the manifest
- Updating inventory records and other relevant data systems

Overall, manifest matching is a critical process that helps to ensure the accuracy and efficiency of logistics and supply chain operations and can provide valuable insights into the performance of these operations over time.

2.3 Dynamic Configurations

Allow developers and system administrators to manage and update application settings more efficiently, without requiring downtime or interruption of the application. This is particularly useful in environments where applications need to be able to adapt quickly to changing conditions, such as in cloud computing environments or in applications that need to be highly available. Dynamic configurations also help to reduce the risk of configuration errors or mistakes, as configuration parameters can be easily managed and updated through a centralised interface, rather than requiring manual editing of code or configuration files.

There are several key benefits of using dynamic configurations:

1. Agility: Dynamic configurations enable applications to be more agile and flexible, allowing them to adapt quickly to changing requirements or conditions.

2. Scalability: Dynamic configurations can be easily scaled to accommodate growing or changing application requirements, without requiring significant changes to the underlying code.

3. Reliability: Dynamic configurations help to ensure that application settings are up-to-date and accurate, reducing the risk of configuration errors that can lead to system downtime or other issues.

4. Security: Dynamic configurations can help improve security by allowing sensitive configuration parameters to be managed separately from the application code and stored in a secure database or file.

Overall, dynamic configurations are an important design pattern in modern software development and can help organizations build flexible and adaptable applications that can respond quickly to changing requirements or conditions, while also improving agility, scalability, reliability, and security.

The Matcher application did not have dynamic configurations. The configurations were hosted as part of the service code itself. So, any change in the configuration needs the redeployment of the whole service.

2.4 Data Warehousing

Data warehousing is the process of collecting, storing, and managing data from various sources in a centralised repository. The goal of data warehousing is to provide a single, unified view of an organisation's data, which can be used for reporting, analysis, and decision-making.

Data warehousing involves a number of processes, including data extraction, transformation, and loading (ETL), as well as data modeling and schema design. Data is typically stored in a structured format, such as a relational database, and is optimized for reporting and analysis.

Data warehousing is often used in large organizations or businesses that have complex data requirements, as it can help to improve data quality, reduce redundancy, and provide a consistent view of data across the organization. Data warehouses can also be used to store historical data, which can be useful for trend analysis and forecasting.

One of the key benefits of data warehousing is that it can help organizations to gain insights into their business operations and improve decision-making. By providing a comprehensive view of data, organizations can identify patterns and trends that may not be apparent in individual data sources and can use this information to make more informed decisions.

Overall, data warehousing is an important technology in the field of data management and can be a powerful tool for organizations that need to manage and analyze large amounts of data.

Currently, the data warehousing is performed on the data enriched by the existing Ingestion platform as part of the matching process. This is done by transforming the enriched data into an easily consumable format and publishing it into the particular locations of the s3 buckets from where the downstream consume the data for records and analysis.

2.5 Summary

In this chapter, the related work is discussed with their purposes and the current implementation methods of the different features of the Matching component of the Ingestion platform.

Chapter 03: SYSTEM DESIGN

Managing a project, whether it is about software development or planning a corporate event, can be daunting, to say the least. The reality is that there is no secret formula that will make your project unwrap flawlessly; very often you'll stumble upon a series of challenges and obstacles before you reach success. And the ability to overcome those unexpected challenges and deal with them on the go is what makes a successful software developer.

This chapter discusses the system analysis and design for the project and discusses the technical terms that will be engaged throughout the whole project.

3.1 System Analysis

The problem statement describes a service that is accomplished through multiple tasks. Being a team project, the tasks of the service are distributed among the team members. This analysis will cover the tasks that the author has been assigned to.

The following tasks were worked on and completed by the author in this project to date:

- Dynamic Configurations: The configurations of the service were defined in a static file. Any change in the configuration needs deployment of the service to take effect. To get rid of the deployments required for change in the configurations, the concept of dynamic configurations came into the picture.

- Data Warehousing: The service being built enriches the matching results that need to be stored and published for the downstream to consume for record and analysis.

3.1.1 Dynamic Configuration

Dynamic configurations can be used in various contexts, such as application configuration settings, system behavior settings, or feature toggles. They may include parameters such as database connection strings, API endpoints, logging levels, caching settings, or other runtime settings that affect the behavior or performance of a software system.

The benefits of using dynamic configurations include:

1. Flexibility: Dynamic configurations allow for easy and quick changes to system settings without requiring a system restart or redeployment. This can enable faster response to changing business requirements, operational needs, or user preferences.

2. Adaptability: Dynamic configurations allow for systems to adapt to different environments or conditions, such as adjusting performance settings based on system load or changing integration settings based on external factors.

3. Maintainability: Dynamic configurations can simplify the management and maintenance of software systems by centralizing configuration settings and providing a single point of control for making changes.

4. Testing and Deployment: Dynamic configurations can facilitate testing and deployment processes by allowing different configurations to be used in different environments, such as development, testing, staging, and production, without requiring code changes or recompilation.

One of the most popular solutions for this problem is AWS AppConfig. With AWS AppConfig, users can create configurations for their applications, such as database connection strings, API endpoints, and other settings, and store them in a centralized configuration store. These configurations can be updated in real-time, without the need to restart the application or make any code changes.

AWS AppConfig also provides feature flagging capabilities, which allow users to control the rollout of new features to their applications. By using feature flags, users can release new features to a subset of users, test them, and gradually roll them out to more users. This can help reduce the risk of releasing new features and improve the overall quality of the application.

Some of the key benefits of using AWS AppConfig for dynamic configurations include:

1. Flexibility: AWS AppConfig supports various types of configuration data, including strings, JSON objects, and YAML files, which can be customised based on the specific needs of the application.

2. Scalability: AWS AppConfig can handle configurations for applications running on thousands of servers, providing a scalable solution for managing configurations.

3. Security: AWS AppConfig integrates with AWS Identity and Access Management (IAM), allowing users to control access to their configuration data based on IAM policies. 4. Real-time updates: Configurations can be updated in real-time, without the need to restart the application or make any code changes, allowing for more agile development and faster deployment.

3.1.2 Data Warehousing

Data warehousing is a process of collecting, organising, and analysing large amounts of data to support business intelligence and decision-making processes. It involves the creation of a central repository or database that integrates data from various sources and transforms it into a format that can be easily analysed and queried.

The data warehouse is designed to support complex queries and data analysis, allowing business users to identify patterns, trends, and insights that are not readily apparent in individual data sources. It typically includes historical data that can be used to track trends over time and make more informed decisions about future operations.

Data warehousing involves several key processes, including data extraction, transformation, and loading (ETL), data modelling, and data analysis. The ETL process involves extracting data from various sources, transforming it into a common format, and loading it into the data warehouse. Data modelling involves designing the structure of the data warehouse to support efficient and effective data analysis. Data analysis involves querying and analysing the data to generate insights and support decision-making processes.

AWS Lambda seems the best possible solution to create the Data Warehousing solution for this project. We will be using AWS S3 as the data warehouse. Using Lambda directly to load the data will create excessive writes on S3 which makes the process more costly. To reduce this cost, we introduce an intermediate stream that will buffer before writing to the data warehouse. This will ensure consistent writes to the data warehouse reducing the write costs.

3.2 Design

The right design ideas and methods provide the necessary road map for addressing program complexity and scalability. Additionally, the task of creating this roadmap is given to seasoned software developers and system design strategists with the necessary training. A sound system design strategy necessitates foresight and a deeper comprehension of both the present and future needs of the software product.

The system, as discussed in the Section 3.1 is divided into two sub-modules or subsystems. Figure 3.2 shows the relation and flow of data between the two subsystems or phases.

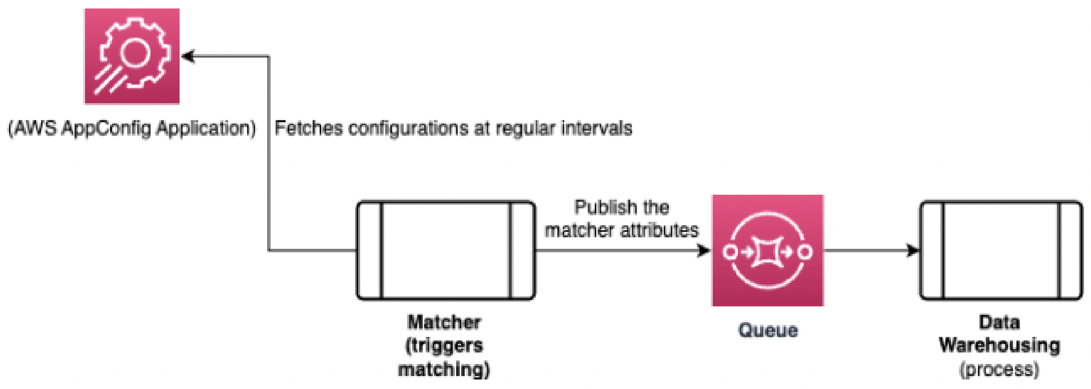
(AWS AppConfig Application)
Fetches configurations at regular intervals
Publish the matcher attributes
Matcher (triggers matching)
Queue
Data Warehousing (process)

Figure 3.1: System Architecture

The designs for the different modules of the proposed system are as follows:

3.2.1 Design for Dynamic Configurations

Figure 3.2.1 shows the basic high-level design for implementing the dynamic configurations setup in the Matcher.

The Matcher application has a client setup called the AppConfigCachingClient that has a TTL configuration, which basically tells the client to refresh the configurations at that interval. Whenever a change in the remote configuration is made, the configuration caching client defined in the application gets the new configuration as it keeps syncing the configuration from the source. This eliminates the need for deployment of the service for changing the configurations of the application.

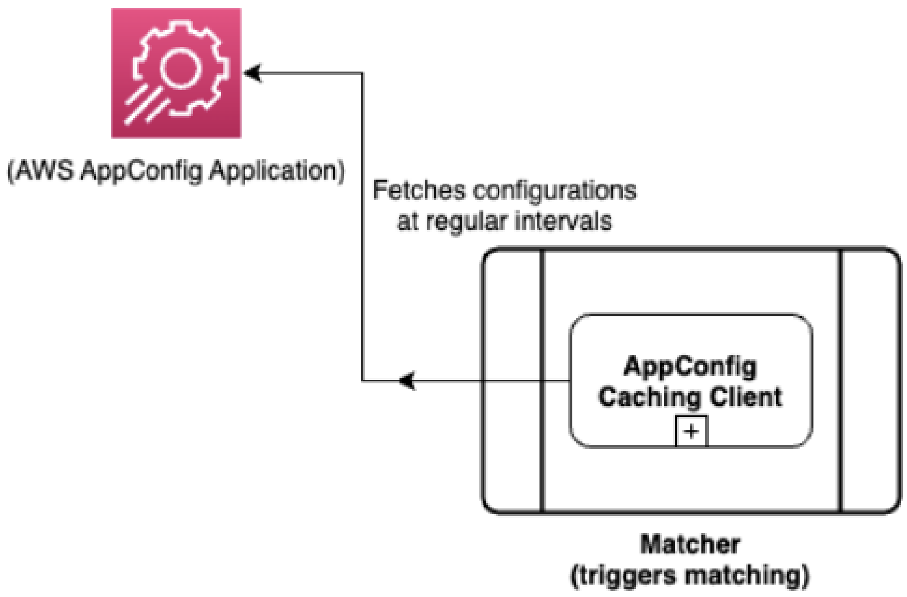
(AWS AppConfig Application)
Fetches configurations at regular intervals
AppConfig Caching Client
Matcher (triggers matching)

Figure 3.2: Dynamic Configurations

3.2.2 Design for Data Warehousing

Figure 3.2.2 shows the basic flow of data in the process of data warehousing. The basic processes that are part of the DW, abbreviated as ETL (Extraction, Transformation, and Loading) are:

1. Data Extraction: This part is done by the Lambda that gets triggered whenever a message is published to the queue. The lambda extracts the data and processes it.
2. Data Transformation: This part of the DW is taken care of by the lambda handler. The lambda handler is responsible for the transformation of the extracted data into the required reporting format.
3. Data Loading: This part of the DW is taken care by the lambda and the firehose. The lambda pushes the reporting data to the firehose delivery stream, which is basically a stream that has some flush configuration. It has a destination, where the delivery stream writes the records at some intervals as defined by the stream flush configuration or bufferingHints property.

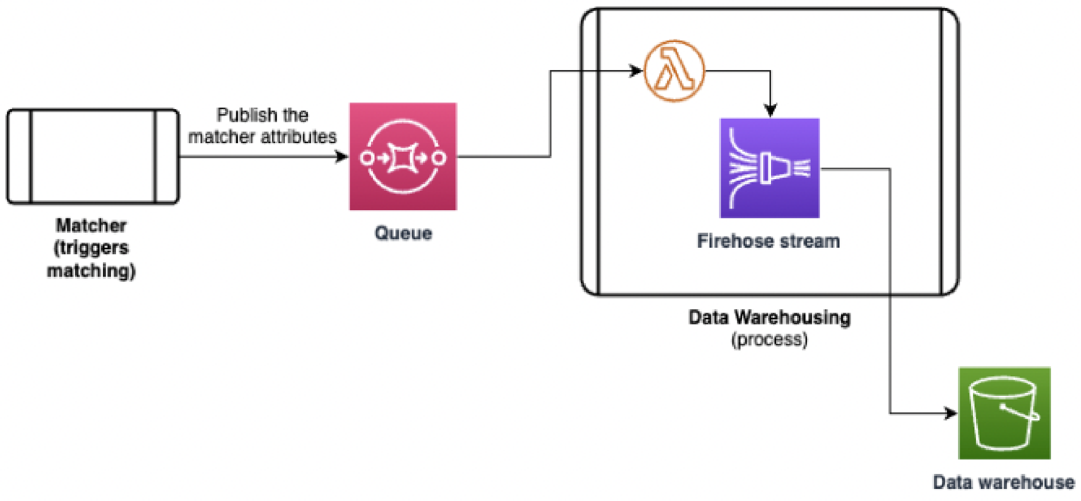
Matcher (triggers matching)
Publish the matcher attributes
Queue
Firehose stream
Data Warehousing (process)
Data warehouse

Figure 3.3: Data Warehousing

3.2.3 Technologies Used

The service is implemented in the "Java Spring framework". The service is built, and deployed on the AWS infrastructure.

The following is the list of AWS Resources that are widely used in the service:

- AWS CDK
- AWS Servers
- AWS Resources

1. **AWS CDK**: AWS CDK apps use the AWS CDK to specify AWS infrastructure and may be created in TypeScript, JavaScript, Python, Java, C#, or Go. One or more stacks are defined by an app. Stacks, which are comparable to AWS CloudFormation stacks, contain constructs. Each construct defines one or more specific AWS resources, such as Lambda functions, Amazon DynamoDB tables, or Amazon S3 buckets. The CDK Toolkit (commonly known as the CLI), a command line tool for interacting with your AWS CDK apps and stacks, is a component of the AWS CDK. The toolkit offers the ability to perform the following among other things:
    a. Convert one or more AWS CDK stacks to AWS CloudFormation[2] templates and related assets (a process called synthesis)
    b. Deploy your stacks to an AWS account and Region

Figure 3.4: Technologies Used in this project

2. **AWS Servers**: AWS servers refer to the virtual machines (VMs) that are provisioned by AWS for customers to run their applications and workloads on. AWS offers several types of servers, each optimised for specific use cases and workloads. Some of the commonly used AWS server types include:
   - Amazon Elastic Compute Cloud (EC2): EC2 is a virtual server that can be used to run a variety of applications and workloads. It provides customers with complete control over their computing resources.
   - Amazon Elastic Container Service (ECS): ECS is a container management service that makes it easy to run, stop, and manage Docker containers on a cluster.

- Amazon Elastic Kubernetes Service (EKS): EKS is a managed Kubernetes service that makes it easy to deploy, manage, and scale containerized applications on Kubernetes.
- AWS Lambda: Lambda[6] is a serverless computing service that allows developers to run their code without having to provision or manage servers.
- Amazon Lightsail: Lightsail is a simplified virtual private server (VPS) that is designed for developers, students, and small businesses. It provides an easy-to-use interface and pre-configured virtual servers.

3. AWS Resources: The following AWS Resources are mostly used in the service:
    a. AWS SQS
    b. AWS S3
    c. AWS Kinesis Data Firehose
    d. AWS SNS
    e. AWS DynamoDB
    f. AWS IAM

3.3 Summary

In this chapter, the discussion is made around the importance of proper planning of a project and how this planning affects the progress of the project. Dividing the problem into different phases and planning it with some timeline comes under the system analysis. Proper analysis will lead to destiny, whereas improper analysis will lead to the failure of the project. The analysis part discusses the different tasks to achieve in the project and what those tasks will achieve. Finally, the design part of the project is laid out describing in brief the technologies and libraries used, and the data flow diagrams of the different sub-modules.

Chapter 04: IMPLEMENTATION

By implementing a project that relies on the strategic planning described earlier in the process, a team can achieve the project's goals while staying within budget and meeting key deadlines. Implementation is the part of a project cycle that links planning with project outcomes. How well this process step is executed ultimately determines the outcome of a project.

In this chapter, the proposed features of Dynamic Configurations & Data Warehousing are briefly discussed. The implementation steps of these features are discussed in this chapter.

4.1 Implementation Dynamic Configurations

Dynamic Configurations are often used in modern software development and cloud computing environments, where the goal is to build scalable, flexible, and resilient systems that can be easily maintained and updated. By allowing configurations to be updated dynamically, developers can make changes to the system in real-time, which can help to reduce the risk of system failures or downtime.

Dynamic configurations can be implemented using a variety of techniques, such as using configuration files that can be edited at runtime, using a centralised configuration server that can be updated on the fly, or using containerization technologies such as Docker, which allow for the creation of lightweight, portable applications that can be easily deployed and updated.

In this project, we implement dynamic configurations using the service provided by AWS which is AWS AppConfig. AWS AppConfig enables the deployment, management, and updating of application configurations. It allows developers to define configuration settings for their applications and deploy them to different environments, such as development, testing, and production.

With AWS AppConfig, developers can manage configurations for their applications without having to redeploy the application or make code changes. It supports dynamic configurations, which means that application settings can be changed in real time without any downtime or disruption to the application.

AWS AppConfig provides several features, such as versioning, validation, and deployment capabilities, that help developers manage configurations efficiently. It also integrates with other AWS services, such as Amazon S3 and AWS Lambda, to provide a complete solution for managing application configurations.

The steps involved in implementing this part involve:

1. Create a new AWS CDK application for hosting AppConfig.
2. Setup the environments
3. Define the strategy to host the configuration profiles.
4. Setup the AppConfig Stack and add the required configurations in any format (JSON, YAML, YML, etc.)
5. Deploy the AppConfig stack in the AWS account. Once the deployment is successful, the application can be seen in the AWS Management Console. The application consists of the environment and the configuration profiles hosted in that environment.
6. In the service, setup the AppConfigCachingClient which fetches the configurations from the AWS AppConfig and catches them for some time defined by TTL in seconds at the application level.
7. Write the parsers for the various types of configurations added in the above steps for parsing the fetched configuration to make it consumable by the various flows of the service.
8. Make sure the application has enough permissions to access the AppConfig resource. In the absence of permissions, the application will not be able to access the hosted configurations. The permissions can be managed using the AWS IAM roles.

The following are the benefits of this implementation:

- Centralised management: With AWS AppConfig, you can manage your application configurations centrally from a single console, making it easier to update and deploy configurations across multiple environments.
- Improved application availability: By separating configuration data from application code, AWS AppConfig makes it easier to update and deploy configurations without having to redeploy the entire application, which can improve application availability.
- Faster deployments: AWS AppConfig can help you deploy configuration changes faster by enabling you to roll out changes in a controlled and gradual manner. This can help reduce the risk of deployment errors and ensure a smoother transition.
- Reduced operational costs: By automating the deployment and management of application configurations, AWS AppConfig can help reduce operational costs and increase operational efficiency.
- Increased security: AWS AppConfig supports encryption of configuration data, making it easier to secure sensitive information such as API keys and passwords.
- Integration with other AWS services: AWS AppConfig integrates with other AWS services such as Amazon CloudWatch and AWS Systems Manager, making it easier to monitor and manage application configurations alongside other AWS resources.

4.2 Implementation of Data Warehousing

Data warehousing is a process of collecting, storing, and managing large volumes of data from various sources to support business intelligence (BI) activities such as data analysis, reporting, and decision-making. The data is typically extracted from transactional databases, operational systems, and external sources, and then transformed and loaded into a central repository called a data warehouse.

Data warehousing involves several steps, including data extraction, data transformation, data loading, and data querying. The extracted data is transformed into a format that is suitable for analysis and is then loaded into the data warehouse. Once the data is loaded into the warehouse, it can be queried and analysed using various BI tools, such as reporting and visualisation tools, to gain insights into business operations and performance.

AWS Lambda can be used for several data warehousing tasks, such as ETL (Extract, Transform, Load) processes, data transformations, data enrichment, and data processing. AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS), which allows users to run code without having to manage servers or infrastructure.

With AWS Lambda, users can write custom code in various programming languages, including Python, Java, and Node.js, and deploy it as functions. These functions can be triggered automatically by other AWS services, such as Amazon S3 or AWS Glue when new data is available or when certain events occur.

One of the key benefits of using AWS Lambda for data warehousing is the ability to scale quickly and efficiently. With AWS Lambda, users can create functions that can scale automatically to handle large volumes of data, without having to to manage servers or infrastructure. This makes it an ideal solution for data warehousing tasks that require scalability, such as ETL processes and data transformations.

Another benefit of using AWS Lambda for data wcomputinging is the cost-effectiveness of the service. With AWS Lambda, users only pay for the compute time that the use their function, which can result in significant cost savings compared to traditional server-based solutions.

Data warehousing with AWS Lambda nd Firehose can provide a complete and powerful solution for managing, processing, and analyzing data. AWS Lambda is a serverless computing service that allows users to run code in response to events, while Amazon Kinesis Firehose is a fully managed service that makes it easy to load streaming data into AWS data stores and analytics tools.

Using AWS Lambda and Firehose together, users can build scalable and efficient data warehousing solutions that can handle large volumes of data in real time.

Here’s how it works:

1. Data is ingested into Firehose from various sources, such as IoT devices, social media, or other sources.
2. Firehose automatically batches and compresses the data and then delivers it to a destination, such as Amazon S3 or Amazon Redshift.
3. As the data is delivered to the destination, it triggers a Lambda function that performs data transformations, data enrichment, or other data processing tasks.
4. The processed data is then loaded into a data warehouse or other analytics tools, where it can be analyzed and visualized.

By using AWS Lambda and Firehose together, users can achieve several benefits, including

1. Scalability: AWS Lambda and Firehose can scale automatically to handle large volumes of data, without having to manage servers or infrastructure.
2. Real-time processing: With Firehose, data can be delivered to the destination in near real-time, allowing for real-time analytics and insights.
3. Cost-effectiveness: Users only pay for the compute time that their Lambda functions use, and Firehose offers a cost-effective solution for delivering data to AWS data stores and analytics tools.
4. Flexibility: AWS Lambda supports various programming languages, allowing users to write custom code for data processing tasks.

Based on the sampling approach, the data warehousing solution has been designed for the Matcher application. The attributes that Matcher enriches for the invoices need to be published downstream for the record and analysis. The below steps include the implementation of Data Warehousing for Matcher as per the design proposed in the previous chapter:

1. Create the data warehouse bucket in the AWS S3, Kinesis Data Firehose delivery stream that uses the S3 bucket as the destination.
2. Setup a Java Lambda application and write the lambda handler functions to perform

the extraction and transformation of the reporting data, and to push the transformed data to the Firehose.

3. Setup the Lambda function with SQS as the event source and handler as the function defined in the previous step. Setting the SQS as the event source for lambda will enable the triggering of the lambda function whenever a message is received in the SQS queue.
4. Deploy the lambda application for the lambda function to pick up the definition.
5. Start publishing the matcher attributes to the queue.

After the setup is complete and deployed in the required AWS accounts, it will work the following way:

1. When a message is received in the SQS queue, the lambda function will be triggered.
2. Lambda will get the message from the queue, perform the required transformation and push the final record to the Firehose delivery stream.
3. Firehose delivery stream will periodically flush the stream data to the destination S3 bucket.

4.3 Summary

This chapter discusses the implementation of the two major tasks of the project. The design and the implementation are done taking all the factors into consideration like cost, efficiency, ease of scaling, etc. to ensure the developed systems are accurate and cost-effective.

Chapter 05: Result Analysis and Testing

When we use analytics, we want to learn something valuable from the process, therefore it's critical that the data gathered and used actually has value. At its best, analytics can result in significant discoveries and opportunities, but at its worst, it can squander time and put an excessive reliance on potentially deceptive data.

In a software development cycle, the value of software testing and quality assurance is significant. Both procedures improve the entire process and guarantee the highest possible product quality.

Software Testing Advantages:

1. Cost-Effective: Software testing has a number of advantages, one of which is affordability. Any IT project can save you money in the long run if testing is done in a timely manner. Fixing faults is less expensive if they are discovered earlier during the software testing process..
2. Security: This delicate and risky benefit of software testing. People are looking for trustworthy products. It helps to eliminate risks and problems earlier.
3. Product Quality: Product quality is a crucial need for every software product. Customers will always obtain a high-quality product thanks to testing.
4. Customer satisfaction: Every product has as its main objective to satisfy clients. Testing for UI/UX ensures the best possible user experience.

At Amazon, testing takes place at different levels. We make sure to cover all the possible edge and corner cases so that the developed system works as ideal systems in the production offering the customers the best service. We have multiple non prod environments to test in, each with different kinds of configuration. As we move towards the prod environment, the test environment configurations are designed in a way that the environment get closer to what the prod environment will be.

The following types of testing are performed on any developed system before taking it to production (some kind of testing may not be applicable on some system):

1. **Unit Testing**: A software testing technique known as unit testing involves testing each individual unit or component of a software system separately in order to ensure that each one performs as intended. A function, method, or class can be a unit.

   Unit testing is used to make sure that each piece of code carries out its intended function accurately and that the integration of these pieces of code functions as planned. Typically automated, unit tests are created to find flaws and faults early in the development cycle, when fixing them is simpler and less expensive..

   Some of the benefits of unit testing include:
   - Early detection of defects: Unit testing can detect errors and defects early in the development process, which can save time and money by avoiding more expensive fixes later on.
   - Improved code quality: By testing each unit of code in isolation, developers can ensure that their code is functioning as intended and that it meets the required specifications.
   - Better maintainability: Unit tests can help ensure that changes to the codebase do not introduce new errors or regressions, making it easier to maintain and modify the code over time.
   - Faster development cycles: Unit testing can help speed up the development process by catching errors early, reducing the time spent on debugging and testing.

   Some best practices for unit testing include:
   - Writing tests that are independent of each other: Tests should be written so that they do not depend on each other, which can help ensure that each test is testing a specific functionality and that it is not affected by other tests.

- Testing both positive and negative cases: To make sure that the code reacts effectively to both expected and unexpected conditions, such as invalid input, unit tests should test both positive and negative scenarios.
- Testing edge cases: Unit tests should test edge cases, such as the minimum and maximum input values, in order to ensure that the code handles these cases correctly.
- Automating tests: Unit tests should be automated in order to ensure that they are run consistently and that they can be easily integrated into a continuous integration/continuous deployment (CI/CD) pipeline.

2. **Integration Testing:** The purpose of integration testing is to ensure that the interactions between different components of a software system are working as expected and that the system as a whole meets the required specifications. Integration testing can be particularly important in complex software systems, where individual components may work correctly in isolation but may fail when integrated with other components.
   Some of the benefits of integration testing include:
   - Improved quality: Integration testing can help improve the overall quality of a software system by detecting defects and errors that may only appear when different components are integrated with each other.
   - Reduced risk: By identifying and addressing issues in the integration of different components early in the development process, integration testing can help reduce the risk of problems arising later in the development cycle or after release.
   - Better communication: Integration testing can help identify communication issues between different components, which can improve communication and collaboration between development teams working on different components.
   - Enhanced reliability: Integration testing can help ensure that a software system is reliable and performs as expected in different environments and under different conditions.

   Some best practices for integration testing include:

- Planning and design: Integration testing should be planned and designed carefully, with a clear understanding of the components that need to be tested, their dependencies, and the expected results.
- Test environment: Integration testing should be performed in an environment that simulates the production environment as closely as possible, in order to identify and address issues that may arise in real-world scenarios.
- Test data: Test data should be carefully selected and designed to ensure that it covers all the possible scenarios that may occur during integration testing.
- Automation: Integration testing should be automated whenever possible, in order to reduce the time and effort required and to ensure consistency in the testing process.

3. **Load Testing**: Load testing involves testing a software system under heavy loads to evaluate its performance and behaviour under high stress conditions. The goal of load testing is to identify and eliminate performance bottlenecks, and to ensure that the system can handle the expected user load without degrading in performance or crashing.
   Load testing is typically performed by simulating a large number of concurrent users or requests, and by measuring the system's response time, throughput, and resource utilisation. Load testing can be done for different types of software systems, including web applications, APIs, and databases.

   Some of the benefits of load testing include:
   - Improved system performance: Load testing can identify performance bottlenecks in a software system, and can help developers optimise the system for better performance under high loads.
   - Enhanced reliability: Load testing can help identify issues related to system stability and reliability, and can help ensure that the system can handle heavy loads without crashing or losing data.
   - Better scalability: Load testing can help ensure that a software system can scale up or down as needed to handle changes in user load or other demands.

- Improved user experience: Load testing can help ensure that a software system provides a good user experience, even under high loads.

Some best practices for load testing include:

- Planning and design: Load testing should be planned and designed carefully, with a clear understanding of the expected user load, the test scenarios, and the expected results.
- Test environment: Load testing should be performed in an environment that simulates the production environment as closely as possible, in order to identify and address issues that may arise in real-world scenarios.
- Test data: Test data should be carefully selected and designed to ensure that it covers all the possible scenarios that may occur during load testing.
- Automation: Load testing should be automated whenever possible, in order to reduce the time and effort required and to ensure consistency in the testing process.

5.1 Result Analysis

The test results of this system for different types of testing are as follows:

1. Unit Testing Results: We intend to cover the different possible scenarios to make sure the certain piece of code is working fine. To achieve this, first we check the test coverage report. The goal is to cover 100% of the lines, methods and branches. After the coverage report gives 100% coverage, we proceed on to test with different input and output scenarios.
2. Integration Testing Results: The integration tests were written keeping in mind the best practices discussed above. The integration tests passed with the expected output produced.
3. Load Testing is not applicable for the features covered as part of this report. Load tests are usually performed where the load of traffic can affect the efficiency of the service.

5.2 Summary

In this chapter, we discussed the importance of result analysis and testing. The test results of the system and its developed features ensure the best quality and efficiency.

Chapter 06: CONCLUSION

A successful project is the one which keeps on getting updated and upgraded with the new features, latest frameworks and upgraded versions of itself. The manifest matching project has successfully achieved its objectives of automating the invoice validation process and improving the efficiency of the supply chain operations. By integrating the manifests with the invoice validation system, the project has reduced the error rates and minimised the need for manual intervention. The project has also enabled the organisation to streamline its operations, optimise its resources, and enhance its overall productivity. The findings from this project have significant implications for the industry and can contribute to the broader goals of improving the accuracy and reliability of supply chain operations. As we reflect on this project, we recognize that there are opportunities for further improvement and refinement. We recommend that the organisation continues to invest in the development of the manifest matching system, leveraging emerging technologies and best practices to enhance its functionality and scalability.

Bibliography

[1] https://docs.aws.amazon.com/cdk/v2/guide/home.html

[2] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

[3] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

[4] https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html

[5] https://docs.aws.amazon.com/eks/latest/userguide/index.html

[6] https://docs.aws.amazon.com/lambda/latest/dg/index.html

[7]https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html

[8] https://docs.aws.amazon.com/AmazonS3/latest/userguide/index.html

[9] https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html

[10] https://docs.aws.amazon.com/sns/latest/dg/index.html

[11] https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/index.html

[12] https://docs.aws.amazon.com/IAM/latest/UserGuide/index.html

Appendix A

Amazon Leadership Principles

Amazon Leadership Principles are a set of guiding values and beliefs that Amazon uses to hire, evaluate, and develop its employees. These principles are considered fundamental to Amazon's culture and are used to drive decision-making and behaviour at all levels of the organisation.

The Amazon Leadership Principles are as follows:

1. Customer Obsession: Leaders start with the customer and work backwards. They work vigorously to earn and keep customer trust. Although leaders pay attention to competitors, they obsess over customers.
2. Ownership: Leaders are owners. They think long term and don't sacrifice long-term value for short-term results. They act on behalf of the entire company, beyond just their own team. They never say "that's not my job."
3. Invent and Simplify: Leaders expect and require innovation and invention from their teams and always find ways to simplify. They are externally aware, look for new ideas from everywhere, and are not limited by "not invented here." As we do new things, we accept that we may be misunderstood for long periods of time.
4. Are Right, A Lot: Leaders are right a lot. They have strong judgement and good instincts. They seek diverse perspectives and work to disconfirm their beliefs.
5. Learn and Be Curious: Leaders are never done learning and always seek to improve themselves. They are curious about new possibilities and act to explore them.
6. Hire and Develop the Best: Leaders raise the performance bar with every hire and promotion. They recognize exceptional talent and willingly move them throughout the organisation. Leaders develop leaders and take seriously their role in coaching others.
7. Insist on the Highest Standards: Leaders have relentlessly high standards – many people may think these standards are unreasonably high. Leaders are continually raising the bar and driving their teams to deliver high-quality products, services, and processes.

8. Think Big: Leaders create and communicate a bold direction that inspires results. They think differently and look around corners for ways to serve customers.
9. Bias for Action: Leaders act quickly and decisively. They make progress every day and are comfortable with ambiguity and risk. They seek to simplify and are internally driven to deliver results.
10. Frugality: Accomplish more with less. Constraints breed resourcefulness, self sufficiency, and invention. There are no extra points for growing headcount, budget size, or fixed expense.
11. Earn Trust: Leaders listen attentively, speak candidly, and treat others respectfully. They are vocally self-critical, even when doing so is awkward or embarrassing. Leaders do not believe their or their team's body odour smells of perfume. They benchmark themselves and their teams against the best.

These principles are used by Amazon to evaluate job candidates, provide feedback to employees, and make business decisions. They are considered a critical part of Amazon's success and are integral to its corporate culture.

ORIGINALITY REPORT

11%	6%	3%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to National School of Business Management NSBM, Sri Lanka Student Paper	1%
2	getnave.com Internet Source	1%
3	Submitted to University of Wales Institute, Cardiff Student Paper	1%
4	Submitted to RMIT University Student Paper	<1%
5	Submitted to University of Zululand Student Paper	<1%
6	docs.aws.amazon.com Internet Source	<1%
7	ebin.pub Internet Source	<1%