Final Internship Project

A Project Report submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Madarapu Sree Pramod (112115080)

Under the Supervision of: Mr Vikas Goyal

Semester: VIII

Department of Computer Science and Engineering

Indian Institute of Information Technology, Pune

(An Institute of National Importance by an Act of Parliament)

May 2025

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Final Internship Project"** submitted by **Madarapu Sree Pramod** bearing the **MIS No: 112115080** in completion of his project work under the guidance of **Mr Viaks Goyal** is accepted for the project report submission in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in the **Department of Computer Science and Engineering**, Indian Institute of Information Technology, Pune (IIIT Pune), during the academic year **2024-25**.

Mr Vikas Goyal
Project Supervisor
Senior Manager Software Engineering
IIIT Pune

Dr. Bhupendra Singh
Head of the Department
Assistant Professor
Department of CSE
IIIT Pune

Project Viva-voce held on _____

Undertaking for Plagiarism

I **Madarapu Sree Pramod** solemnly declare that the research work presented in the report titled **"Final Intership Project"** is solely **my** research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete report has been written by **me**. I understand the zero tolerance policy of **Indian Institute of Information Technology, Pune** towards plagiarism. Therefore **I** declare that no portion of my **report** has been plagiarized and any material used as reference is properly referred/cited. I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of the degree, the Institute reserves the rights to withdraw/revoke my **B.Tech** degree.

Madarapu Sree Pramod

14/05/25

Conflict of Interest

Manuscript title: Final Internship Project

The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Madarapu Sree Pramod

14/05/25

ACKNOWLEDGEMENT

This project would not have been possible without the help and cooperation of many. I would like to thank the people who helped me directly and indirectly in the completion of this project work.

First and foremost, I would like to express my gratitude to our honourable Director, Prof. O.G. Kakde, for providing his kind support in various aspects. I would like to express my gratitude to my project guide, **Mr Vikas Goyal, Senior Manager, Software Engineering**, for providing excellent guidance, encouragement, inspiration, and constant and timely support throughout this B.Tech Project. I would like to express my gratitude to the Head of Department, **Dr. Bhupendra Singh**, **Department of CSE**, for providing his kind support in various aspects. I would also like to thank all the faculty members in the **Department of CSE** and my classmates for their steadfast and strong support and engagement with this project.

.

Abstract

Internship Role: Software Development Engineer (SDE) Intern
Organisation: WM Global Technology Services India Pvt. Ltd

During my internship a**t Walmart Global Tech,** I contributed to diverse tasks, enhancing my technical expertise and practical knowledge. I was part of the development team for **Growth and Monetise** Vertical, which is responsible for attracting and increasing traffic to **Walmart's e-commerce websites**. Specifically, I have worked on components related to **Display Ads,** which are hosted on various Websites and web pages. A significant example is the ads presented when a Web Browser User uses **Google search,** which show relevant products related to their query. My primary objective was to make modifications in the way the data stores used for this task are created to accommodate the needs of my team. I have also helped in the testing of the new Marketing Assistant UI that is in development. This UI will be used for the streamlining of the above process of sending data to partners, which involves the use of Hive and BigQuery tables. I have worked on various diverse technologies in this project including but not limited to technologies like **Git** and **GitHub, Python, Java**, **Scala**, with Testing moudules like **JUnit** and **Mockito**, along with services like **Google Cloud Project(GCP)** for its Storage and analytical capabilities, **Workflows** and **Astonomer** for handling **Airflow** DAGs, **Spark** Jobs for execurting the workflows and Walmarts internal solutions for CI/CD pipelining and deployments. I have also worked extensively on manipulating **Hive** and **BigQuery** tables for processing and extracting large amounts of data.

Keywords: Display Ads, Git and GitHub, Java, Python, Scala, JUnit, Mockito, GCP, CI/CD Pipelines, Workflows, Airflow, Astronomer, Spark, Hive, BigQuery, Maven.

.

TABLE OF CONTENTS

.

Chapter 1
Introduction

1.1. Overview of work

During my internship at WM Global Technology Services India Pvt. Ltd, I have taken the role of Software Developer Intern and was assigned the task of developing a new approach for the publishing of Product and Offer data to the end Partners. Partners refer to entities that display product Ads on their platforms. This new approach includes the use of a Marketing Assistant UI(MA UI), which is designed for scheduled transfers of data using Airflow DAGS. An Airflow DAG is used for creating and executing a pipeline, which in this case is used as a data pipeline from WM to third-party partners. The data for this is available in the Marketing Assistant Platform(MAP) BigQuery Table. This data undergoes several transformations in the middle before finally being formatted into a Hive Table, which we will call the Publish Data Table(PDT). Then Hive SQL is used to extract certain fields from this PDT according to the requirements of the partners, and the data is sent to a specified external file location from where the Partner can access the data. This task has to be performed every day since WM needs to send the latest information, which will be displayed in the Ads.

The former approach for this task of data publishing included the use of conventional Java and Scala classes and objects, which are used to build a component called as Publishers. An Executable JAR was created from these classes, which contained the entire logic for extracting, performing some processing on the extracted data and then writing it to a location where the partner can access it. This data is taken from the MAP BQ table and not from the PDT.

1.2. Literature Review

The internship project focused on optimising data pipelines for delivering products and offering information to advertising partners. This involved leveraging various technologies and following best practices in data engineering and workflow orchestration.

a. Data Warehousing and Analytics

Apache Hive is a data warehouse infrastructure built on top of Hadoop, enabling the querying and analysis of large datasets using a SQL-like language. It allows for scalable and efficient data summarisation, which is crucial for processing extensive product catalogues.

Google BigQuery is a fully managed, serverless data warehouse that allows for super-fast SQL queries using Google's infrastructure's processing power. It supports real-time analytics

.

and is optimised for handling massive datasets, making it suitable for analysing user interactions and ad performance.

b. Workflow Orchestration

Apache Airflow is an open-source platform designed to programmatically author, schedule, and monitor workflows. It uses Directed Acyclic Graphs (DAGs) to manage task dependencies, ensuring that complex data pipelines execute in the correct order. Airflow's extensibility and integration capabilities make it a preferred choice for orchestrating data workflows.

Astronomer provides a managed service for Apache Airflow, offering enhanced scalability, monitoring, and deployment options. It simplifies the management of Airflow instances, allowing for more efficient workflow orchestration in production environments.

c. Programming Languages and Tools

Python is widely used in data engineering for scripting, automation, and developing data processing applications. Its simplicity and extensive libraries make it ideal for building and managing data pipelines.

Java and Scala are robust, high-performance languages commonly used in big data processing. They are particularly effective when working with distributed data processing frameworks, enabling the development of scalable and efficient data transformation applications.

Version control systems like Git and platforms such as GitHub are essential for collaborative development, code management, and maintaining the integrity of codebases. They facilitate continuous integration and deployment (CI/CD) practices, ensuring that updates to data pipelines are systematically tested and deployed.

d. Testing and Quality Assurance

JUnit is a widely used testing framework for Java applications, allowing developers to write and run repeatable tests to ensure code reliability. Mockito complements JUnit by enabling the creation of mock objects for unit testing, which is crucial for isolating components and verifying their behaviour independently.

e. Discussion on Former Approach

As discussed, the former approach included a group of Scala classes and objects which used a Proto Buffer schema to keep track of the data, which is stored as JSON objects. The structure of a product record involved details regarding the product and the offer currently assigned to that product. The Protocol Buffer class is used to create a “.proto” file, which is used for storing the schema of the JSON object, through which it becomes easy and efficient to extract

the data. These collections of Scala classes contain some special classes which are specific to each Partner. So, for every new partner, Scala classes have to be defined, which will extract and process data according to the needs of the Partner. These requirements may differ for each Partner, so a general class cannot be created for all.

The JAR of this collection of classes is built and executed using Airflow DAGs. A workflow is created for the JAR, which stores configuration and run informaton about how to run the main method of the JAR. A workflow can contain the rules of executing multiple classes in the JAR, and the order in which these classes have to be executed can be configured when creating the workflow. This workflow is then run using Spark jobs in Kafka clusters.

Now, there is a development for a new approach where the data extraction and processing is shifted to a UI based approach using Hive SQL and the PDT instead of the MA BQ table.

1.3 Motivation of Work

The primary motivation behind this project was to enhance the efficiency and scalability of Walmart's data publishing system to external advertising partners. The existing method relied on a set of Scala classes and objects, utilising Protocol Buffers to structure product and offer data. This approach required the creation of partner-specific Scala classes to handle varying data requirements, leading to increased development time and maintenance complexity.

The new approach aimed to address these challenges by introducing a more flexible and maintainable system. By leveraging a Marketing Assistant UI (MA UI) for scheduling data transfers and utilising Apache Airflow DAGs for workflow orchestration, the process became more streamlined. Data is now extracted from the Marketing Assistant Platform (MAP) BigQuery Table, transformed, and stored in a standardised Publish Data Table (PDT) within Hive. This allows for dynamic data extraction using Hive SQL, tailored to partner requirements without the need for additional Scala class development.

This transition not only reduces development overhead but also enhances the system's adaptability to accommodate new partners and changing data needs. The motivation was to create a more robust, scalable, and efficient data publishing pipeline that aligns with modern data engineering practices.

1.4 Research Gap

The research gap here can be thought of as the difference in the processing happening before the storing of product and offer data in the PDT. The difference in this processing between the Publishers and the PDT table gives some trouble in utilising the MA UI and Hive SQL directly. So the difference in the processing has to be established for each of the attributes that are used, and changes have to be made. These changes can happen either in the creation logic of the PDT or in the Hive SQL used for extracting the published data. The challenges and

benefits of both these methods have to be analysed, and the complexities of the solution also need to be considered. Since Walmart deals with a large volume of data every day, any loss of efficiency can have a large effect on the time to perform these tasks.

The PDT is also used by many internal teams, so the consequences of making modifications to the populating logic can have a large impact on them. So all these elements need to be considered before taking any approach.

.

Chapter 2

Problem Statement

Walmart's existing product-and-offer data publishing pipeline relies on custom Scala "Publisher" modules and partner-specific Protocol Buffer schemas to extract, transform, and format data for each advertising partner. This tightly coupled, code-centric approach has led to a proliferation of nearly identical classes, one for every partner.. Onboarding a new partner or updating an existing feed requires manual class authoring, JAR builds, Airflow DAG configuration, and end-to-end testing across multiple environments, resulting in onboarding cycles that can stretch from days to weeks. These challenges underscore the need for a flexible, UI-driven solution that centralises data transformations in Hive SQL, standardises schema management via a single Publish Data Table, and empowers non-technical stakeholders to configure, preview, and schedule partner feeds without modifying underlying code.

2.1. Analysis and Design

2.1.1 Functional Requirements

a. Partner Feed Configuration

- Allow users to select one or more partners and define their specific data needs (fields, filters, output format).
- Persist each partner's configuration for reuse and auditability in the UI.

b. Data Extraction & Transformation

- Read from the standardised Publish Data Table (PDT) in Hive, containing the merged product and offer Subjson records.
- Apply partner-defined filters and field selections via HiveQL.
- Support incremental (daily) execution to fetch only new or changed records.

c. Scheduling & Orchestration

- Enable users to assign schedules (daily/weekly) for each partner feed.
- Leverage Apache Airflow DAGs—managed in Astronomer—to orchestrate extract → transform → load steps.
- Surface DAG run status and logs in the UI.

d. Delivery & Monitoring

- Write query results to partner-specific file locations (GCS buckets, SFTP, etc.).
- Provide basic monitoring and alerting on failures or data-volume anomalies.

.

2.1.2 Non-Functional Requirements

- **Scalability:** Handle growing numbers of partners and daily data volumes (tens of millions of records).
- **Usability:** Intuitive, self-service UI that non-technical users can employ without engineering support.
- **Maintainability:** Minimal code changes required when adding new partners or data fields.
- **Reliability:** Automated retries and fallbacks in Airflow; end-to-end testing pipelines.
- **Security:** Access controls on the UI, DAGs, and data stores; encrypted data transfers.

2.1.3 Technology Stack

- **Main Logic for Building and Populating PDT and MA UI BQ Table:**
    - Scala
    - Java with **Maven** for building JARS.
    - Python for automation
    - Hive SQL for Table Operations
- **Data Processing:** Apache Hive, HiveQL, Parquet
- **Workflow:** Apache Airflow (Astronomer)
- **Cloud Services:** Google Cloud Storage, BigQuery, Cloud Workflows
- **CI/CD:** GitHub Actions, Internal CI/CD pipelines
- **Testing:** JUnit, Mockito, Jest (for UI)

.

Chapter 3
Proposed Work

The first solution we explored was to directly modify the processing logic used during the creation of the Publish Data Table (PDT). Because the PDT feeds many downstream internal teams, any change would ripple across multiple services and require formal review and approval before production rollout. My initial assignment was to dive into the GitHub repository housing the PDT-creation code, pinpoint the relevant modules, and map out how each piece contributed to the final Hive table. This meant tracing through several Scala source files—both standard classes and case class definitions that represent the product and offer schemas—and understanding how they tie together.

Within the codebase, I identified all the necessary code files and classes. There are classes for creating intermediary tables—one for product records and one for offer records—each populated by running parameterised Hive SQL queries against the Marketing Assistant Platform's BigQuery table. These queries extract raw JSON data, parse it via Protocol Buffer (.proto) definitions, and materialise it into typed Scala objects using the schema defined in Case classes for both tables. From there, the Scala logic transforms these objects as needed (e.g., handling nulls, normalising date formats) and writes the results as Parquet files to a designated Google Cloud Storage bucket.

Then these files are used for creating the PDT by performing a Join Operation on them using a Key. This logic is present in a different Scala file, and the process used HIVE SQL and manipulation of partitions.

Building and packaging these components rely on Apache Maven. The pom.xml configuration not only manages dependencies (Scala, Hadoop, Protobuf libraries) but also injects environment-specific settings, such as GCS bucket paths, Hive metastore URIs, and BigQuery project IDs through Maven profiles. By switching profiles, the same JAR artifact can target development, staging, or production clusters without requiring code changes.

Orchestration of the entire workflow is defined in Google Cloud Workflows configurations. Each workflow entry specifies which Scala class to invoke, the sequence of calls (for example, extract-product → extract-offer → join-and-write), and command-line arguments (such as the date partition or partner identifier). These YAML-style definitions serve as a single source of truth for how the JAR should run in each environment, and they feed directly into the internal CI/CD pipeline to automate deployments.

Finally, Astronomer-managed Apache Airflow takes over for day-to-day execution. We maintain a template Airflow DAG that reads the appropriate Cloud Workflow configuration

For each run, it triggers the workflow and monitors its progress. Alerts are configured to surface any failures or performance anomalies. This layered approach—code logic in Scala, environment orchestration via Maven and Cloud Workflows, and scheduling through Airflow—provides clear separation of concerns but also underscores why a UI-driven, SQL-first method could dramatically simplify both development and operations.

These executions are run using Spark Jobs whose configuration is defined in the code files. They run in internally managed Kafka Clusters. Unit Testing was also performed for the above approach using Junit and Mockito.

However, this approach was later revised, and I had to create a new PDT table that would not impact the internal teams' operations. This new table has specific filters which are useful in extracting only the data that will be needed for the Publish data. This approach has optimised the time taken due to the less data that needed to be processed.

In this approach, I had to make changes to the code and accommodate the creation of a new table from the already existing code without writing new classes for this new table. This involved creating new and independent intermediate products and offering tables, and then a new PDT table with specific filters.

Maven arguments were used to differentiate between the different table creation logics, and If-Else blocks were used to choose which table to create or use at forks. Unit Tests were written wherever needed using Junit and Mockito. A new Workflow was created, and a successful run has been made.

I am now in the process of creating a Publish data and testing it against the Data produced by the old code-based uploader.

3.1 Methodology of work

The methodology for modifying Walmart's data-publishing pipeline unfolded in three main phases: (1) initial analysis of the existing PDT creation logic, (2) implementation and validation of a first "in-place" modification approach, and (3) the design and rollout of an isolated, partner-focused PDT solution. Each phase is described in detail below:

Phase 1: Repository Exploration & Requirements Gathering

1. **Clone & Explore Codebase**

   - Checked out the GitHub repository containing the Scala code and resource files for PDT generation.

   - Mapped out the directory structure and classes of different uploaders

- Identified key Scala classes and case class schemas corresponding to Protocol Buffer definitions.

2. **Document Data Flow**

- Traced how raw JSON rows from the MAP BigQuery table flow through Hive-SQL scripts, Protobuf parsing, Scala transformations, and Parquet serialisation.
- Noted each step's inputs and outputs, including intermediate products and offer tables and final PDT partitions.

Phase 2: First Modification Approach

1. **Schema Adjustments in Scala**

- Introduced minor changes to the existing case class and populate method definitions to accommodate new partner fields.
- Updated Hive-SQL scripts used for columns and filter logic.

2. **Build Configuration with Maven**

- Added new Maven profile flags (e.g. -PmodifyPDT) to toggle between old and modified schema logic.
- Used resource filtering in pom.xml to inject GCS paths, Hive metastore URIs, and BigQuery project IDs for each environment.

3. **Workflow & Orchestration Updates**

- Modified existing Cloud Workflows YAML to pass the new flag to the JAR entry point.
- Updated the Airflow DAG template in Astronomer to reference the revised workflow for nightly runs.

4. **Unit Testing & Validation**

- Wrote JUnit and Mockito tests for new parsing and transformation logic.
- Ran Spark jobs on the development Kafka cluster to generate test Parquet files and compared record counts against expectations.

.

5. **Review & Roadblocks**

- Gathered feedback from internal teams; realised schema changes risked breaking multiple consumers and required lengthy approval cycles.

Phase 3: Isolated Partner-Focused PDT Solution

1. Designing a Parallel Table

- Decided to create a new, independent PDT variant filtered for partner needs, leaving the original PDT untouched.
- Defined filter criteria (e.g., only active products, specific product types) to reduce data volume and speed up processing.

2. Code Refactoring Without New Classes

- Reused existing Scala modules but parameterised them: introduced IF-ELSE logic in that checks a Maven flag to determine which intermediate tables and final table to build.
- Configured new Hive-SQL snippets for the partner-specific joins and filters, stored alongside the originals.

3. Workflow Creation & Scheduling

- Authored a new Cloud Workflows to invoke the JAR with the flag and appropriate date and partner ID arguments.
- In the process of adding them to Astronomer as an Airflow DAG.

4. Comprehensive Testing

- Enhanced unit tests to cover both code paths (original vs. partner-specific) using JUnit and Mockito.
- Performed end-to-end runs in a staging environment:
    - Generated both original and partner PDTs side by side.
    - Validated record counts, schema conformance, and Parquet file integrity.
    - Conducted comparison queries in Hive and spot-checked sample outputs.

.

3.2.1. Hardware & Software specifications

Hardware Specifications:

Development and Testing were performed on local systems along with cloud-based architecture.

Development Machine

- **Model:** Apple MacBook Pro (M1 Pro)
- **CPU:** Apple M1 Pro (8-core CPU, 14-core GPU)
- **RAM:** 16 GB unified memory
- **Storage:** 500 GB SSD
- **Operating System:** macOS Sequoia (15.x)

Google Cloud Project was used for Storage, Executing BigQuery SQL quiers and running the Spark jobs in Kafka clusters

Software Specifications

1. **Development Environment**
    - **Code Editors/IDEs:**
        - **Intellij IDEA Ultimate (Scala plugin)**
        - Sublime Text
        - PyCharm IDE for Python scripts
    - **Version Control:**
        - Git with GitHub for repository hosting and branch protections
2. **Programming Languages & Frameworks**
    - **Scala:** 2.11.x (data-processing modules)
    - **Java:** OpenJDK 1.8 (Scala runtime)
    - **Python:** 3.8 (scripting, small ETL tasks)
    - **SQL:** HiveQL (Hive) and Standard SQL (BigQuery)

.

3. **Data Technologies**

- **Apache Hive:** 3.1.2 (data warehousing; Publish Data Table)
- **Apache Spark:** 2.4.5 (executing Scala jobs in Kafka clusters)
- **Protocol Buffers:** protobuf-java 3.12.0 (schema definitions)

4. **Workflow Orchestration & CI/CD**

- **Apache Airflow:** 2.1.0 (managed via Astronomer runtime 0.25.0)
- **Astronomer:** Hosted service for Airflow scheduling and monitoring
- **Google Cloud Workflows:** YAML-based orchestration of JAR executions

- **CI/CD Tools:**
  - GitHub Actions (build, test, deploy pipelines)
  - Walmart's internal CI/CD for production deployments

5. **Build & Dependency Management**

- **Apache Maven:** 3.6.3 (profiles for dev/staging/prod)

6. Testing Frameworks

- **JUnit 5 & Mockito 3.x:** Unit tests for Scala/Java code

3.3 Dataset Description

- Google Cloud was used to store the Hive and BigQuery tables needed for the project.
- The task I was assigned needed the manipulation and modification of several Hive tables and BQ tables.
- All of these datasets contain a large number of records that had to be processed, generally in the tens of millions of records at a time.

.

Chapter 4
Results and Discussion

4.1 Results

- **Improved Pipeline Efficiency:**
  The redesigned workflow noticeably accelerated daily data processing, reducing the time required to generate and deliver partner-specific feeds. By filtering at the source and working with a leaner dataset, the overall resource usage was more balanced and consistent.

- **Streamlined Partner Onboarding:**
  Introducing a UI-driven configuration eliminated the need to write custom code for each new partner. This shift turned a previously time-intensive onboarding process into a straightforward configuration task, allowing new feeds to go live in a fraction of the time.

- **Enhanced Reliability:**
  Standardised SQL templates and a single, authoritative Publish Data Table dramatically cut down on workflow errors. Automated retries and clear alerting ensured that failures were rare and, when they did occur, were quickly detected and resolved.

4.2 Discussion

Transitioning from bespoke Scala modules to a metadata-driven, SQL-first approach proved transformative. By centralising transformation logic in Hive SQL and exposing it through a guided UI, the team achieved both greater agility and stronger governance. Engineering effort shifted away from routine code changes toward building more advanced features and addressing edge-case transformations.

That said, highly complex enrichment steps still rely on custom code. Future enhancements could explore integrating user-provided scripts or UDFs into the UI workflow, broadening the range of partner requirements that can be handled without backend changes. Additionally, versioned configurations and rollback capabilities would further bolster operational resilience.

Overall, this project underscored how a balance of self-service tooling and robust, standardised data models can drive both efficiency and innovation in large-scale enterprise environments.

.

Chapter 5

Conclusion and Future Scope

Conclusion

The migration to a UI-driven, SQL-first data-publishing pipeline represents a significant evolution in how Walmart Global Tech delivers product and offer information to our advertising partners. By centralising transformation logic in Hive SQL and exposing it through the Marketing Assistant UI, we have dramatically reduced the complexity and turnaround time for partner onboarding, while improving reliability and governance. Throughout the project, I played a dual role—enhancing the backend Publish Data Table logic and actively collaborating with the MA UI team on functional and load testing. This hands-on involvement ensured that the UI not only meets technical requirements but also delivers a seamless, intuitive experience for non-technical users. As we validate the new PDT table and the MA UI in staging, early feedback confirms that the solution is both robust and extensible.

Future Scope of the project

- **Full UI Rollout:** Complete end-to-end validation in production and decommission legacy Scala pipelines for partners who have migrated.
- **Pushing the new PDT creation code to production.**
- **Performance Tuning:** Continue load-testing under peak conditions and optimise Hive and Airflow configurations to maintain low latencies as data volumes grow.

.

References

[1] Apache Hive™ Documentation. (2024). “Welcome to Apache Hive.” Retrieved from https://hive.apache.org/

[2] Google Cloud. (2024). BigQuery Documentation. Retrieved from https://cloud.google.com/bigquery/docs

[3] Google Cloud. (2024). Cloud Storage Documentation. Retrieved from https://cloud.google.com/storage/docs

[4] Google Cloud. (2024). Cloud Workflows Documentation. Retrieved from https://cloud.google.com/workflows/docs

[5] Astronomer. (2024). “Astronomer Airflow.” Retrieved from https://www.astronomer.io/docs

[6] Apache Airflow™ Documentation. (2024). “User Guide.” Retrieved from https://airflow.apache.org/docs/apache-airflow/stable/

[7] Protocol Buffers. (2024). “Protocol Buffer Language Guide.” Retrieved from https://protobuf.dev/reference/protobuf/lang/de

[8 Scala Documentation. (2024). “Scala 2.12 Documentation.” Retrieved from https://docs.scala-lang.org/overviews/2.12/overview.html

[9] Apache Maven™ Documentation. (2024). “Introduction to the POM.” Retrieved from https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

[10] GitHub. (2024). “GitHub Docs: Managing Branches.” Retrieved from https://docs.github.com/en/repositories/configuring-branches-and-merges

[11] JUnit 5 User Guide. (2024). “Writing Tests with JUnit 5.” Retrieved from https://junit.org/junit5/docs/current/user-guide/

[12] Mockito Documentation. (2024). “Mockito Official Site.” Retrieved from https://site.mockito.org/

.

A Note to Whomever It May Concern

Respected Sir/Ma’am,

Subject: Request for Consideration of Internship Completion Documents Submission
I hope this message finds you well.

I am writing to inform you about the status of my internship, which commenced on **15th of January**
and will conclude on 13**th of June**. As my internship is still ongoing, I am currently unable to produce the final internship certificate. I kindly request your understanding regarding this matter and
assure you that I will provide the internship completion certificate as soon as my internship concludes.

Thank you for your support and understanding.

Warm regards,

Madarapu Sree Pramod,
9246690006,

.