Finance Receipting using Robotic Process Automation

A PROJECT REPORT

Submitted by,

Manoj J - 20211CSE0577
Mohammed Maaz Rehman - 20211CSE0612
Anjan GM – 20211CSE0611

Under the guidance of,

Dr. Prasad P S

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

At

PRESIDENCY UNIVERSITY BENGALURU

JANUARY 2025

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project report **"Finance Receipting using Robotic Process Automation"** being submitted by Manoj J, Mohammed Maaz Rehman, Anjan GM bearing roll number(s) 20211CSE0577, 20211CSE0612, 20211CSE0611 in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a Bonafide work carried out under my

supervision.

Dr. PRASAD P S
Assistant Professor-
Selection Grade,
PSCS
Presidency University

Dr.ASIF MOHAMMED H B
Head of Department,
School of Engineering
PSCS
Presidency University

Dr. L. SHAKKEERA
Associate Dean
PSCS
Presidency University

Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-Vc School of Engineering
Dean -PSCS-IS
Presidency University

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work, which is being presented in the project report entitled **Finance Receipting using Robotic Process Automation** in partial fulfilment for the award of Degree of **Bachelor of Technology** in Computer Science and Engineering is a record of our own investigations carried under the guidance of **SUPERVISOR Dr. Prasad P S**, **Assistant Professor – Selection Grade, School of Computer Science Engineering & Information Science, Presidency University, Bengaluru.**

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

Manoj J (20211CSE0577)
Mohammed Maaz Rehman (20211CSE0612)
Anjan G M (20211CSE0611)

ABSTRACT

Automation of a back-office process that has 25000 receipts in a month. The data is structured with predefined rules and conditions. The process involves 10% exceptions and uses paper. Source file is downloaded from the SAP application, where the data is pulled from. Client details are separated into ups, cash, cheque, online banking, or card based on transaction type. Segregated user information is fed into the web application. For each Receipt number, templates are used to generate receipts and the same is mailed to the vendor specified in the source file. Each transaction has its own template type, which is available in the File Server.

ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, ProVC, School of Engineering and Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Deans **Dr. Shakkeera L and Dr. Mydhili Nair,** School of Computer Science Engineering & Information Science, Presidency University, and Dr.Asif Mohammed, Head of the Department, School of Computer Science Engineering & Information Science, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr.Prasad P S, Assistant Professor**-Selection Grade and Reviewer **Ms.Bhuvaneshwari Patil, Assistant Professor**, School of Computer Science Engineering & Information Science, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work.

We would like to convey our gratitude and heartfelt thanks to the PIP2001 Capstone Project Coordinators **Dr. Sampath A K, Dr. Abdul Khadar and Mr. Md Zia Ur Rahman,** department Project Coordinators Mr. Amarnath J and Dr. Jayanthi K and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

Manoj J

Mohammed Maaz Rehman

Anjan GM

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER-1
INTRODUCTION

In today's fast-paced business environment, automation has become a crucial tool for streamlining repetitive and labor-intensive tasks. This is especially true for back-office operations, such as financial receipting, which require high accuracy and efficiency. With organizations processing large volumes of transactions daily, manual processing not only slows down operations but also increases the risk of human error.

Our client, a large enterprise, has 25,000 receipts to process monthly. What makes it complicated is that some of these are in digital and paper forms, apply predefined rules, and have a 10% exception rate. The current process is to download receipt data from an SAP application, separate the transactions into Card and Cheque type, create customized receipts on each type of transaction using pre-defined templates stored in file servers, and then send these receipts through email to the relevant vendor as listed in the source file**.**

The goal of this project is to design an RPA solution that automates the entire process of receipting. This RPA solution retrieves data from SAP, applies business rules, generates receipts based on a template, and sends receipts to vendors, along with proper handling of exceptions, reducing processing time to a minimal amount, increasing the accuracy level, and ensuring it follows the defined business rules in all aspects.

CHAPTER-2
LITERATURE SURVEY

2.1 Manual Finance Receipting:

Human judgment and flexibility: Can adjust immediately and deal with complex cases.No technological dependency: Simple processes do not require investments in software or hardware. Error-prone Human errors are highly probable because of the repetitive and manual data entry. Time-consuming: Slow processing speed, especially for high volumes (e.g., 25,000 receipts per month). Expensive: Requires a large workforce as volume increases, leading to higher operational costs. Hard to scale: Difficult to handle increasing workloads without proportionally increasing staff.

2.2 Basic Script-Based Automation (e.g., Excel Macros, VBA Scripts)

Easy to establish: It demands very little technical knowledge and could be established readily using inhouse tools such as Excel. Low cost: No need for advanced automation software, which reduces the cost, it requires for small and simple processes. Less flexible: Difficult to alter the script in complex business rules or massive amounts of data. Bad scalability: Scripts are not developed to manage the volume and complexity at an enterprise level. Tough maintenance: The script must be changed every time the process changes, hence the more the maintenance time and effort is consumed.

2.3 Robotic Process Automation

Highly scalable: Built to process millions of transactions efficiently without hassles, like the 25,000 receipts generated each month. Error reduction: Automates lengthy processes with zero room for human error.

Cross-platform integration: Compatible with diverse systems, including SAP, email servers, and other enterprise platforms without hitch. Structured exception handling: Deals with the pre-defined exceptions very efficiently, and it requires human intervention only for the most complex cases.

Initial investment is too high: Involves investments in RPA platforms such as UiPath and Automation Anywhere. Complex setup: Building strong automation workflows might require technical expertise. Ongoing maintenance: If business rules or systems change, then monitoring and updating must be continuously done.

2.4 OCR with RPA

Processing physical paper documents: OCR converts the paper receipts into digital. They can be processed like digital receipts. Data extraction speed: This kills the need to manually enter data into any system since it reads and extracts text from scanned images. Accuracy: It is better with structured data. For instance, structured forms where the layout of receipts is standard. Inaccuracies with unstructured documents: OCR may fail with low-quality or layout-varied documents and need to be manually verified. High initial cost for integration: Integrating OCR with RPA is more complex and increases the cost and complexity. Not error-proof: Though OCR makes things efficient, it still brings in some errors, especially when paper or scanned receipts are involved.

2.5 Enterprise Resource Planning (ERP) System Integration (e.g., SAP)

Centralized data management: ERP systems like SAP can handle finance receipting end-to-end using built-in modules. Consistency and compliance: assures the corporate policies can be followed along with data integrity at all departments and units

Better reporting: Helps provide real-time data and analytics, thus managing to make faster decisions. Limited flexibility: The ERPs are designed for rigid processes, and so any automation process however simple or complex turns into complicated and expensive development. High cost: The implementation, customization, and maintenance of ERP systems are costly, especially for a specific use case like receipting. Steep learning curve: The use of the system requires training and expertise, slowing down immediate implementation.

2.6 BPM Tools:

Holistic approach: BPM tools provide a more comprehensive view of business processes, optimizing not only automation but the entire workflows. Flexibility They allow

for customization and adaptation as business rules change, integrating various business processes like invoicing, receipting, and more.

Visual workflows: Often come with visual tools for easy process mapping and automation. High complexity: BPM solutions involve detailed process mapping and substantial time for design and implementation. High cost: These tools are expensive to license and maintain. Not specific to finance: Although BPM tools support a variety of business processes, they might not be specific enough for specific finance tasks, such as the complex exception handling involved in receipting.

2.7 Artificial Intelligence (AI) for Intelligent Automation Benefits:

Sophisticated exception handling: AI can process dynamic scenarios and complex exceptions using predictive analysis and machine learning models.

Learning with time: AI models improve with increasing data processing and will need less human intervention when handling exceptions. Scalable and adaptable: AI may adapt to various types of documents, learning from interactions that bring incrementally more correct answers.

2.8 Cloud-Based Automation Solutions:

Cost-effective: There is no investment in infrastructure, as most cloud-based automation services operate on a pay-as-you-go model. Global accessibility: Cloud solutions enable global teams to access the system and manage processes from any location. Automatic update: Cloud platforms ensure that the software is always up-to date without manual intervention. Security issues: Critical financial information processed in the cloud can be compromised if security is not in place. Latency and performance: There is a risk of delays in processing based on connection and server location, especially in very high-volume intake situations such as receipting. Customization is limited: Cloud-based solutions may not provide on premises options for dealing with custom workflows.

2.9 Workflow Automation Tools (e.g., Zapier, Microsoft Power Automate)

User-friendly: Needs minimal technical knowledge and can be set up with drag-and-drop interfaces. Fast to implement: Enables rapid deployment for simple tasks like sending emails, processing data, or transferring files. Integration with multiple apps: These tools support integration with multiple third-party applications, including web apps and cloud services.

2.10 Custom-Built Automation Solutions:

Tailored to specific needs: Custom solutions can be built exactly to meet the specific requirements of the business, ensuring higher efficiency. Full control over functionality: The company has complete control over the features, integrations, and updates. High development cost: Building custom solutions from scratch is resource intensive and expensive. Longer implementation time: Development can take months or even years, especially for complex workflows.

Ongoing maintenance: Custom solutions have dedicated resources allocated to updates, bug fixes, and modifications since the business needs will change.

CHAPTER-3
RESEARCH GAPS OF EXISTING METHODS

In this process, receipts are downloaded from an SAP application, segregated with respect to payments made through type (Upi, Cash, Cheque, Online Banking, Card), and further customized receipts prepared for each different type of payment using pre-prepared templates found in a file server. Those receipts must, after preparation, be sent out via email to the respective vendors as indicated on the source file.

The objective for this project would be to create and implement a Robotic Process Automation (RPA) solution that automates the entire receipting process. Retrieval of information from SAP systems, application of business rules for generating receipts on templates, as well as mailing receipts to suppliers will be effectively done with adequate handling of exceptions. This automates the task to a larger extent, reduces time consumption, achieves higher accuracy and adherence to previously defined business rules.

CHAPTER-4
PROPOSED METHODOLOGY

4.1 Problem Understanding and Requirement Gathering

- **Goal:** Understand how finance receipting is done currently and what problems exist in the current system which may be either manual or semiautomated.

Activity: 1.

Have discussions with stakeholders to comprehend the entire process of receipting, including receipts volume (25,000 per month), transaction types (Card, Cheque), and why it needs to be automated.

Track the data flow from receipt generation to distribution, including exception cases (10% exceptions). 3. Identify software systems (e.g., SAP) and third-party tools that are used in the current process.

4.2 Process Design and Flowchart Creation

Objective: Design the end-to-end automated process flow so that all steps of finance receipting are covered.

Steps:

- **Data Extraction:** The source file of receipt data is downloaded from SAP. Define how the SAP system exports structured data such as receipt numbers, client details, and transaction types.

- **Data Segregation:** According to transaction type (Card or Cheque), the receipt data is segregated into respective groups.

▪ Use decision rules to classify the data and prepare it for processing.

- **Data Processing:** For each receipt, the appropriate template is chosen from the File Server, depending on the transaction type.

▪ Template mapping should be defined for each transaction type (Card vs Cheque) to create a formatted receipt.

- **Exception Handling:** Identify any missing data or incorrect transaction types that require human intervention.

▪ Exception rules can be used to flag the issue and route it to a user for manual processing.

Automation Flowchart: Create a flowchart illustrating how the automation process will be done starting from data extraction to email sending of the receipt. Also show decision points in the transaction and exception handling.

4.3 Tool Selection and Setup

- **Goal:** Identify and set up the required tools to automate the process, extract data, and send the email.

- **Steps:**
- **RPA Tool Setting:** Install and set up the RPA tool, for example, UiPath, in automation.

▪ Workflow for downloading data from SAP, processing receipts, and sending emails.

- **OCR Integration:** Implement OCR (e.g., Tesseract or ABBYY FineReader) to handle any paper-based receipts. ▪ Configure OCR to digitize receipts, extract key fields (receipt number, client details, etc.), and pass them to the RPA system.

- **Email Automation Setup:** Configure an email server or email automation tool (e.g., SMTP, Outlook) to send receipts to vendors.

▪ Set up email templates with the correct placeholders for vendor information and attach the generated receipts.

4.4 Automation Workflow Development

• **Objective:** Develop the RPA workflow to automate the receipting process.

• **Steps:**

• **Data Import from SAP:**

▪ Automatically retrieve the source file from the SAP system using the RPA tool. It can be scheduled or started manually.

▪ Load data into the RPA environment for further processing.

• **Data Segregation and Validation:**

▪ Logic needs to be built to separate the data based on the type of transaction (Card or Cheque) and validate the data integrity, such as accuracy of client information and valid receipt numbers.

• **Template Selection:**

▪ Define the RPA workflow to identify the correct receipt template from File Server for transaction type.

▪ Fill out the receipt template with the acquired data for a transaction.

• **Exception Handling:**

▪ Incorporate exception handling routines to highlight missing or erroneous data.

▪ Automate routing of flagged exceptions to a human operator via email or a dashboard for manual intervention.

• **Email Dispatch:**

▪ Create the final receipt in the correct format (e.g., PDF) and send it to the appropriate vendor using email automation.

4.5 Testing and Validation

• **Goal:** The system must work correctly, catch exceptions, and produce correct receipts

• **Steps:**

• **Unit Testing:**

▪ Test each workflow individually (e.g., data retrieval, template selection, receipt generation) to make sure they are working as expected.

• **Integration Testing:**

▪ Test the entire workflow integrated together, from data download to email dispatch.

▪ Ensure that the system works for both normal transactions and exception cases.

• **Performance Testing:**

Test the scalability of the system to process huge volumes of receipts, say 25,000 a month, and then measure the speed and accuracy in processing.

• **UAT**: Test the system by end-users or finance/accounting teams in the real world to get feedback to make adjustments accordingly.

4.6. Deployment and Monitoring

Objective: Deploy the final solution and ensure smooth operation in a live environment.

Steps:

• **Deployment:** Deploy the automated solution in the production environment with all the required configurations (for example, server settings, access permissions).

- **Monitoring and Maintenance:** Install monitoring tools to track the performance of the automation system, including the success rate of processed receipts, exceptions, and system errors.

- **Feedback Loop:** Get users to give continuous feedback to improve the system, especially to handle exceptions and for changes in business rules.

CHAPTER-5
OBJECTIVES

➢Designing an automation solution for high-volume finance receipting It should handle 25,000 receipts per month with reduced time and labour spent on manual data entry and processing, thus surpassing the shortcomings of manual and spreadsheet-based methods.

➢Error-free data extraction through OCR and RPA integration

By combining Optical Character Recognition (OCR) with Robotic Process Automation (RPA), the system will digitize and process paper as well as digital receipts correctly, thus resolving the accuracy issues prevalent in OCR systems.

➢ Mechanisms for exception handling to be automated

The automation solution will include intelligent workflows that automatically flag and handle exceptions or route them to human intervention, thus reducing manual oversight and improving overall efficiency, tackling one of the main weaknesses in many automation systems.

➢Develop a centralized and customizable receipting system for transaction management.

By integrating with enterprise systems like SAP and ensuring the solution supports various receipt formats (e.g., Card, Cheque), the system will centralize all receipt-related tasks while offering flexibility in handling different types of transactions.

CHAPTER-6
SYSTEM DESIGN & IMPLEMENTATION

6.1 Problem Understanding and Requirement Gathering

Objective: Understand the process of finance receipting and identify the pain points in the current system (manual or semi-automated).

- **Steps:**

1. Discussion with stakeholders regarding the whole process of receipting, including volumes of receipts (25,000/month), types of transactions (Card, Cheque), and the requirement for automation. 2. Document the data flow from the generation of receipt to its distribution, including exceptions cases (10% exceptions). 3. Identify software systems being used in the current process (e.g.

SAP) as well as any third-party tools.

6.2Process Design and Flowchart Development

- **Objective:** Outline the process flow for a completely automated system to ensure no stone is left unturned at every step involved in finance receipting.
- **Steps:**
- **Data Collection:** The SAP file that downloads receipt data needs to be established, and from it, how the system is set to export structured information about receipt number, client's name, etc.
- **Data Segregation:** Group receipt data according to transaction type, Card or Cheque.

▪ Apply decision rules for grouping and data preparation.

- Data Processing: For every receipt, determine the appropriate template to be picked from the File Server, according to the type of transaction, Card or Cheque.

- Define mapping for each type of transaction, Card vs Cheque, to print a formatted receipt.

- Exception Handling: Identify any missing data or incorrect transaction types that require human intervention.

- Use exception rules to flag issues and route them to a user for manual processing.

- Automation Flowchart: Create a flowchart that outlines the entire automation process from data retrieval to sending the receipt via email. It must include decision points for transaction types and exception handling.

6.3 Tool Selection and Setup

- **Objective:** Choose and configure the required automation tools, data extraction tools, and communication tools.

- **Processes:**

- RPA Tool Configuration: Install and configure the RPA tool (example: UiPath) to automate the process.

- All workflow processes are configured for downloading data from SAP, processing receipts, and sending out emails.

- OCR Integration: This provides integration for OCR, such as Tesseract or ABBYY FineReader, for paper-based receipts.

▪ Integrate OCR functionality to capture and extract receipt digitization and corresponding key fields that would be used to send further into the RPA system for processing. Email Automation Setup - Integrate an email server or an automation tool, example being SMTP and Outlook to ensure that receipts automatically go out to their respective vendors; ▪ Mail templates would already have vendor correct placeholders attached.

6.4 Automation Workflow Development

- **Objective:** RPA Workflow Design for Automated Receipting
- **Steps:**
- **Importing Data from SAP:**

▪ Extract the source file using the RPA tool from the SAP system. The source file may be extracted by either scheduled triggers or manual trigger.
▪ Load the data into the RPA environment to be further processed.

- **Segregation and Validation of Data:**

▪ Logic to separate data as per transaction type, whether it is a Card or Cheque, and data integrity, such as correct client information, valid receipt numbers, etc.

- Template Selection: Configure the RPA workflow to pick up the respective receipt template from the File Server, depending on the transaction type.

▪ Fill the receipt template with the extracted data for each transaction.

- Exception Handling: Include exception handling routines to flag missing or incorrect data.

▪ Automate routes for flagged exceptions to a human operator through email or a dashboard for manual intervention.

• Email Dispatch:

▪ Generate the final receipt in the appropriate format, say PDF and send it to the respective vendor using email automation.

6.5 Testing and Validation

• Objective: The system should work fine, throw an exception, and give a proper receipt.

• **Steps:**

• **Unit Testing:**

▪ Test each of the individual workflow, say, data retrieval, template selection, and receipt generation.

• **Integration Testing:**

▪ Conduct tests with the entire workflow integrated from data download to email dispatch.

▪ Validate that the system works properly for normal transactions as well as exception cases.

•**Performance Testing:** Test if the system will be able to process large number of receipts, let's say, 25,000/month as well as estimate the processing time and accuracy in processing.

•**User Acceptance Testing (UAT):**

Provide the end user of the system in a real-scenario environment from finance or account teams to make its testing and share feedback for tweaking.

6.6 Deployment and Monitoring Objective:

Deploy the solution finally and work smoothly in production.

Steps:

- Deployment: Deploy the automated solution in the production environment, making sure all necessary configurations (e.g., server settings, access permissions) are in place.
- Monitoring and Maintenance: Set up monitoring tools to track the performance of the automation system, including the success rate of processed receipts, exceptions, and system errors.
- Feedback Loop: Obtain user feedback to enhance the system, particularly for handling exceptions and changing business rules.

CHAPTER-7

TIMELINE FOR EXECUTION OF PROJECT

(GANTT CHART)

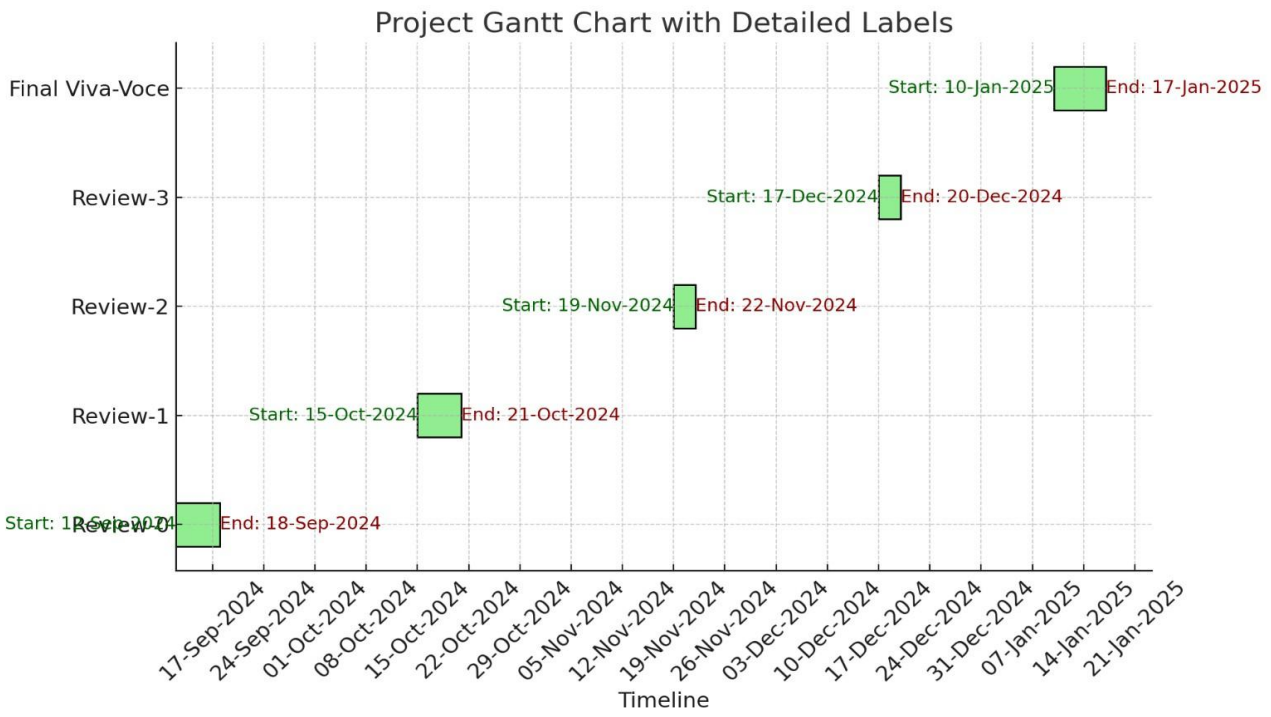


Figure 7.1

CHAPTER-8
OUTCOMES

Better Processing Rate

Result: The automated system is able to process 25,000 receipts per month, which used to take far more time and manual effort under manual or semiautomated processes.

- **Advantage:** Processing time for every receipt came down from several minutes under manual entry to few seconds, making the overall operating efficiency better.

Automated Exception Handling

- **Outcome:** The automation solution processed 90% of the receipts automatically with no human intervention. The remaining 10% of exceptions were identified and directed to the users for further human examination.

- **Benefit:** It allowed the employees to concentrate on the complex exception cases that the automation routine could not handle. In this way, their productivity increased and bottlenecks reduced

Time Savings in Receipt Distribution

- **Outcome:** The system automatically generated receipts using pre-defined templates and emailed them to the respective vendors without any manual intervention, ensuring timely distribution.

- **Benefit:** The time taken to distribute receipts was reduced by 80%, enabling faster and more efficient communication with vendors.

Scalability and Flexibility

- **Outcome:** the RPA solution has shown scalability by the ability to process large quantities of transactions with ease and flexibility in adoption to various types of transactions (Card, Cheque) and changeable business rules.
- **Benefit:** The company is now prepared to take on new growth in transaction volumes without the need for extra manual effort.

Cost Savings

- **Outcome:** The receipting process was automated, and extensive manual labor was reduced, thus saving on operational costs.
- **Benefit:** The company was able to reallocate staff to higher-value tasks and reduce overhead costs associated with manual processing.

Improved Vendor Satisfaction

- **Outcome:** The timely and accurate generation and distribution of receipts led to improved relationships with vendors, as delays and errors were minimized.

- **Benefit:** Vendor queries regarding missing or incorrect receipts decreased, contributing to smoother business operations.

CHAPTER-9
RESULTS AND DISCUSSIONS

The system incorporated Optical Character Recognition (OCR), Robotic Process Automation (RPA), and email automation to facilitate the smooth extraction of data, receipt generation on templates, and dispatch to the vendors in due time. In addition, it demonstrated scalability, being able to manage high volumes of transactions and to have exception management for complicated cases. This led to improved processing time and reduced operational expenses while increasing finance receipting precision, thus positively impacting business continuity and vendor satisfaction. The results demonstrate the project has been successful, meeting the identified goals of being efficient, precise, and highly scalable. Future areas of improvement include fine-tuning exception handling in addition to growing the system so that it allows for more interdependent business flows.

CHAPTER-10
CONCLUSION

The Finance Receipting using Robotic Process Automation (RPA) project addressed the challenges of the manual receipting process. The solution automated the processing of 25,000 receipts per month, thereby significantly improving efficiency and reducing manual errors.

The system ensured seamless data extraction, template-based receipt generation, and timely dispatch to vendors by integrating Optical Character Recognition (OCR), Robotic Process Automation (RPA), and email automation. The solution was scalable, handling large transaction volumes with ease and enabling exception management for complex cases. This reduced processing time, lowered operational costs, and improved accuracy in finance receipting, thus making business operations smoother and vendors more satisfied.

The outcome of this project suggests that the set objectives of efficiency, accuracy, and scalability are successfully accomplished. Future development should concentrate upon fine-tuning the exception handling mechanism and expanding the system's possible capability to include more business processes.

REFERENCES

1. Automation Anywhere. (2023). RPA Best Practices for Finance.
   Retrieved from https://www.automationanywhere.com

2. Blue Prism. (2022). Using RPA in Financial Operations.
   Retrieved from https://www.blueprism.com

3. UiPath. (2023). RPA in Finance: Automating Invoice Processing.
   Retrieved from https://www.uipath.com

4. SAP SE. (2023). Integrating SAP with RPA for Streamlined Finance Operations.
   Retrieved from https://www.sap.com

5. ABBYY. (2022). OCR for Finance: Enhancing Accuracy in Document Processing.
   Retrieved from https://www.abbyy.com

6. Deloitte. (2023). RPA in Finance and Accounting: A Guide.
   Retrieved from https://www2.deloitte.com

7. IBM. (2023). RPA for Finance Professionals. Retrieved from https://www.ibm.com

8. EY. (2022). The Role of RPA in Financial Services.
   Retrieved from https://www.ey.com

9. KPMG. (2023). Robotic Process Automation in Financial Operations.
   Retrieved from https://home.kpmg

APPENDIX-A
PSUEDOCODE

Pseudocode for Sending Email with Attachment

1.Define the send_mail function:

- Input parameters: filename (file to attach) and to_emailid (recipient email address).

2. **Set up email credentials:**
   - Define sender_email with your email address. o Define receiver_email with the recipient's email address.
   - Define password with your email account password (use App Passwords if needed).

3. **Prepare the email content:**
   - Create an email subject line: 'Finance Receipting using Robotic Process Automation'. o Define the email body in HTML format with a structured template.

4. **Compose the email:**
   - Initialize a MIMEMultipart object to handle the email content and attachments.
   - Add sender, recipient, and subject headers to the email object.
   - Attach the HTML body using MIMEText.

5. **Attach a file:**
   - Open the specified filename in binary read mode.
   - Create a MIME object for the attachment and read its contents.

- Encode the file in base64 format. ○ Add headers to the attachment for content disposition and filename. ○ Attach the file to the email object.

6. **Send the email using Gmail's SMTP server:**
   - Establish a connection to the Gmail SMTP server (smtp.gmail.com) on port 587.
   - Start TLS to secure the connection.
   - Log in using the sender email and password. ○ Convert the email object to a string.
   - Send the email to the recipient.

7. **Handle errors:**
   - Use a try block to execute the email-sending logic.
   - Catch and display any exceptions that occur.

8. **Output:**
   - Print a success message if the email is sent successfully.
   - Print an error message if an exception occurs.

Pseudo-Code for Processing Source Files

1. **Import Necessary Modules**
   ○ Import Path from pathlib.
   ○ Import customer classes: UPI_Customer,
   CASH_Customer, CHEQUE_Customer,
   ONLINEBANKING_Customer, CARD_Customer.

2. **Define process_source_files Function**
   ○ Inputs:
   - source_path: Path to the directory containing the source .txt files.
   - delimiter: Delimiter used to split data in each file.

3. **Initialize Directory and Files**
   - Create a Path object for source_path.
   - Retrieve all .txt files from the directory.

4. **Initialize Customer List**
   - Create an empty list customers to store customer objects.

5. **Loop Through Each .txt File**
   - For each file_path in the list of .txt files:
     - Open the file and read its content into content.
     - Split the content using the delimiter into split_data_array.

6. **Determine Customer Type** o Extract the customer type from the 11th element of split_data_array and convert it to uppercase.

7. **Create and Append Customer Objects** o Use conditional logic to check the customer_type:
   - If UPI:
   - Create a UPI_Customer object using appropriate elements from split_data_array.
   - Append the object to customers.
   - If CASH:
   - Create a CASH_Customer object using appropriate elements from split_data_array.
   - Append the object to customers.
   - If CHEQUE:
   - Create a CHEQUE_Customer object using appropriate elements from split_data_array.
   - Append the object to customers.

- If ONLINE_BANKING:
- Create an ONLINEBANKING_Customer object using appropriate elements from split_data_array.
- Append the object to customers.
- If CARD:
- Create a CARD_Customer object using appropriate elements from split_data_array.
- Append the object to customers.

Else:

- Print "Invalid Customer Type".

8. **Return Customer List** ○ After processing all files, return the customers list.

Pseudo-Code for Template Selection and PDF Generation

1. **Import Required Modules** ○ Import the functions to generate receipt PDFs for different transaction types:
   - generate_upi_template_reciept_pdf
   - generate_cash_template_reciept_pdf
   - generate_cheque_template_reciept_pdf
   - generate_onlinebanking_template_reciept_pdf
   - generate_card_template_reciept_pdf

2. **Define Function select_template_and_generate_pdf** ○ Inputs:
   - pdf_file_absolute_name: The absolute path and name of the PDF file to be generated.
   - data: A dictionary containing customer transaction data.

3. **Extract Transaction Type** ○ Retrieve the value of customer_transaction_type from the data dictionary and assign it to transaction_type.

4. **Select Template and Generate PDF**

Use conditional logic to determine which receipt template to generate:

If transaction_type is UPI:

Call generate_upi_template_reciept_pdf with pdf_file_absolute_name and data.

- If transaction_type is CASH:
- Call generate_cash_template_reciept_pdf with pdf_file_absolute_name and data.
- If transaction_type is CHEQUE:
- Call generate_cheque_template_reciept_pdf with pdf_file_absolute_name and data. If transaction_type is ONLINE_BANKING:
- Call generate_onlinebanking_template_reciept_pdf with pdf_file_absolute_name and data.
- If transaction_type is CARD:
- Call generate_card_template_reciept_pdf with pdf_file_absolute_name and data.

5. **End Function Execution** ○ The function completes its task after calling the appropriate PDF generation function based on the transaction type.

Pseudo-Code for Sending Email with Attachment

1. **Import Required Libraries** ○ Import smtplib for SMTP email handling.
   - Import MIMEMultipart, MIMEText, MIMEBase, and encoders for constructing email content and handling attachments.

2. **Define Function send_mail** ○ Inputs:
    - filename: Name of the file to attach.
    - to_emailid: Recipient's email address.

3. **Set Up Email Credentials and Settings** ○ Define sender_email and password (use a secure app password for Gmail).
    - Define receiver_email using the provided to_emailid.

4. **Compose Email Content** ○ Initialize a MIMEMultipart object for the email. ○ Set the From, To, and Subject fields in the email header.
    - Create an HTML email body containing a styled welcome message and add it to the email using MIMEText with the html format.

5. **Attach a File to the Email** ○ Open the file specified by filename in binary read mode.
    - Create a MIMEBase object for the attachment and set its payload to the file content.
    - Encode the payload to base64 and add a ContentDisposition header with the filename.
    - Attach the MIMEBase object to the email.

6. **Establish a Secure SMTP Connection** ○ Connect to Gmail's SMTP server (smtp.gmail.com) on port 587.
    - Start a TLS session to secure the connection.
    - Log in using the sender's email and password.

7. **Send the Email** ○ Convert the email object to a string using as_string().
    - Use the sendmail method to send the email from the sender to the recipient.

8. **Handle Errors** ○ Wrap the SMTP process in a try-except block.
    - Print an error message if any exception occurs.

9. **Print Success Message** ○ Display a confirmation message if the email is sent successfully.

Pseudo-Code for Scheduled Task Execution

1. **Import Required Libraries** ○ Import schedule for scheduling tasks.
    - Import time to introduce delays in the loop.
    - Import requests to make HTTP calls.
    - Import datetime to get the current timestamp.

2. **Define the Target URL** ○ Set the URL of the REST endpoint to a variable (url).

3. **Define the Task Function** ○ Function Name: your_task
    - Print a message with the current timestamp to indicate the task execution.
    - Make an HTTP GET request to the specified URL. ○ Print the status code of the HTTP response. ○ Use a try-except block to handle potential errors during the request:
        - Catch exceptions like connection errors and print an error message.

4. **Schedule the Task** ○ Use schedule. Every to set the task (your task) to run at regular intervals (e.g., every 30 seconds).
5. **Run the Scheduler** ○ Enter an infinite loop using while True.
    - In each iteration:
        - Check and run any scheduled tasks using schedule.run_pending.

- Pause the loop for 1 second using time. Sleep to reduce CPU usage.

6. **Handle REST Endpoint** ○ Ensure the defined REST endpoint (http://127.0.0.1:5000/process) is functional and capable of processing the request.

Pseudo-Code for the UPI_Customer Class

1. **Class Definition** ○ Define a class named UPI_Customer to represent customers using UPI transactions.

2. **Initialization Method (__init__)** ○ Define an __init__ method for initializing customer attributes.
   - Accept the following parameters during object creation:
     1. customer_name
     2. customer_email_id
     3. customer_phone_number
     4. customer_address
     5. customer_pincode
     6. product_code
     7. product_description
     8. product_quantity
     9. purchase_date_time
     10. amount
     11. transaction_type
     12. transaction_id
     13. bank_name
     14. payment_amount
     15. upi_transaction_id
     16. debited_from

- Assign these parameters to the corresponding instance variables of the class.

3. **String Representation Method (__str__)**
   - Define a __str__ method to return a readable string representation of the object's attributes.
   - Concatenate and format the following details into a single string:
     - customer_name, customer_email_id, customer_phone_number, customer_address, customer_pincode
     - product_code, product_description, product_quantity, purchase_date_time, amount
     - transaction_type, transaction_id, bank_name, payment_amount, upi_transaction_id, debited_from
   - Return the formatted string.

4. **Example of Usage** o Create an object of UPI_Customer by passing the required parameters.
   - Call the __str__ method to view the formatted details of the customer.

Pseudo-Code for the CASH_Customer Class

1. **Class Definition** o Define a class named CASH_Customer to represent customers using cash transactions.

2. **Initialization Method (__init__)** o Define an __init__ method for initializing customer attributes.
   - Accept the following parameters during object creation:
     1. customer_name
     2. customer_email_id
     3. customer_phone_number
     4. customer_address
     5. customer_pincode

6. product_code
7. product_description 8. product_quantity
8. purchase_date_time
9. amount
10. transaction_type
11. transaction_id
12. cash_amount

- Assign these parameters to the corresponding instance variables of the class.

3. **String Representation Method (__str__)**
   - Define a __str__ method to return a readable string representation of the object's attributes. - Concatenate and format the following details into a single string:
     - customer_name, customer_email_id, customer_phone_number, customer_address, customer_pincode
     - product_code, product_description, product_quantity, purchase_date_time, amount
     - transaction_type, transaction_id, cash_amount - Return the formatted string.

4. **Example of Usage** - Create an object of CASH_Customer by passing the required parameters. - Call the __str__ method to view the formatted details of the customer.

Pseudo-Code for the CHEQUE_Customer Class

1. **Class Definition** o Define a class named CHEQUE_Customer to represent customers using cheque transactions.

2. **Initialization Method (__init__)**
   - Define an __init__ method to initialize the customer attributes.
   - Accept the following parameters:
     1. customer_name
     2. customer_email_id
     3. customer_phone_number
     4. customer_address
     5. customer_pincode
     6. product_code
     7. product_description 8. product_quantity
     8. purchase_date_time
     9. amount
     10. transaction_type
     11. transaction_id
     12. bank_name
     13. account_number
     14. cheque_number
     15. cheque_date
     16. cheque_amount
     17. bank_ifsc
   - Assign each parameter to an instance variable of the class.

3. **String Representation Method (__str__)** ○ Define a __str__ method to return a string representation of the object. ○ Concatenate and format the attributes into a single readable string, including:
    - customer_name, customer_email_id, customer_phone_number, customer_address, customer_pincode
    - product_code, product_description, product_quantity, purchase_date_time, amount
    - transaction_type, transaction_id, bank_name, account_number, cheque_number
    - cheque_date, cheque_amount, bank_ifsc
    - ○ Return the formatted string.

4. **Usage Example** ○ Create an instance of CHEQUE_Customer by providing the required parameters. ○ Use the __str__ method to print the object's details in a readable format.

Pseudo-Code for the ONLINEBANKING_Customer Class

1. **Class Definition**

   ○ Define a class named ONLINEBANKING_Customer to represent customers using online banking transactions.

2. **Initialization Method (__init__)**

   ○ Define an __init__ method to initialize the attributes of an online banking customer.

   - Accept the following parameters:
     1. customer_name
     2. customer_email_id
     3. customer_phone_number
     4. customer_address
     5. customer_pincode

6. product_code
7. product_description 8. product_quantity
8. purchase_date_time
9. amount
10. transaction_type
11. transaction_id
12. account_number
13. ref_number
14. bank_name
15. bank_branch
16. bank_ifsc
17. transaction_amount
18. obtransaction_type
19. transaction_date

- Assign each parameter to an instance variable of the class.

3. **String Representation Method (__str__)** ○ Define a __str__ method to return a string representation of the object. ○ Concatenate and format the attributes into a single readable string, including:
   - customer_name, customer_email_id, customer_phone_number, customer_address, customer_pincode
   - product_code, product_description, product_quantity, purchase_date_time, amount
   - transaction_type, transaction_id, account_number, ref_number, bank_name
   - bank_branch, bank_ifsc, transaction_amount, obtransaction_type, transaction_date
   - Return the formatted string.

4. **Usage Example** o Create an instance of ONLINEBANKING_Customer by providing all the required parameters. o Use the __str__ method to print the object's details in a readable format.

Pseudo-Code for the CARD_Customer Class

1. **Class Definition**

   o Define a class named CARD_Customer to manage customer details related to card-based transactions.

2. **Initialization Method (__init__)**

   o Define an __init__ method to initialize the attributes of the card customer.

   - Accept the following parameters:
     1. customer_name
     2. customer_email_id
     3. customer_phone_number
     4. customer_address
     5. customer_pincode
     6. product_code
     7. product_description 8. product_quantity
     8. purchase_date_time
     9. amount
     10. transaction_type
     11. transaction_id
     12. card_number
     13. transaction_ref
     14. card_type
     15. transaction_amount

Assign each parameter to an instance variable of the class.

3. **String Representation Method (__str__)**

   ○ Define a __str__ method to provide a readable representation of the object. ○ Concatenate and format the attributes into a single string that includes:

   - customer_name, customer_email_id, customer_phone_number, customer_address, customer_pincode
   - product_code, product_description, product_quantity, purchase_date_time, amount
   - transaction_type, transaction_id, card_number, transaction_ref, card_type, transaction_amount

   ○ Return the formatted string.

4. **Usage Example** ○ Create an instance of the CARD_Customer class, passing appropriate values for each attribute. ○ Print the instance or convert it to a string to view the formatted details using the __str__ method.

1. Define function signup_user(params):

- Set up the database connection string (dsn, username, password).
- Try to:
- Connect to the database.
- Create a cursor for query execution.
- Execute the INSERT query to add user info.
- Commit the transaction.
- Return 'true' if successful.
- Handle database errors and print error message if any.
- Handle other general errors and print error message if any. - Finally, close the cursor and connection.

2. Define function validate_user(params):

- Set up the database connection string (dsn, username, password).
- Try to:
- Connect to the database.
- Create a cursor for query execution.
- If parameters exist, execute SELECT query with parameters.
- Commit the transaction.
- If SELECT query, fetch and return results.
- Handle database errors and print error message if any.
- Handle other general errors and print error message if any. - Finally, close the cursor and connection.

3. Define function store_to_database(customers):

- Set up the database connection string (dsn, username, password).
- Try to:
- Connect to the database.
- Create a cursor for query execution.
- For each customer in the customers list:
- Get customer ID.
- Prepare customer details and transaction details parameters.
- Execute INSERT queries for customer, purchase, transaction details, and other types based on transaction type (UPI, CASH,

CHEQUE, ONLINE_BANKING, CARD).

- Commit the transaction.
- Handle database errors and print error message if any.
- Handle other general errors and print error message if any. - Finally, close the cursor and connection.

4. Define function fetch_customers(query):

- Set up the database connection string (dsn, username, password).
- Try to:
- Connect to the database.
- Create a cursor for query execution.
- Execute SELECT query.
- If SELECT query, fetch and return results.
- Handle database errors and print error message if any.
- Handle other general errors and print error message if any. - Finally, close the cursor and connection.

5. Define function fetch_customer(customer_id):

- Set up the database connection string (dsn, username, password).
- Try to:
- Connect to the database.
- Create a cursor for query execution.
- Execute SELECT query for a particular customer using customer_id.
- Based on transaction type, execute further SELECT queries for related transaction details.
- Return customer and transaction details.
- Handle database errors and print error message if any.
- Handle other general errors and print error message if any. - Finally, close the cursor and connection.

6. Define function execute_db_queries():

- Example of defining queries like UPDATE and DELETE.
- Execute and commit queries as needed.
- Handle connection clean-up and closing.

APPENDIX-B
SCREENSHOTS

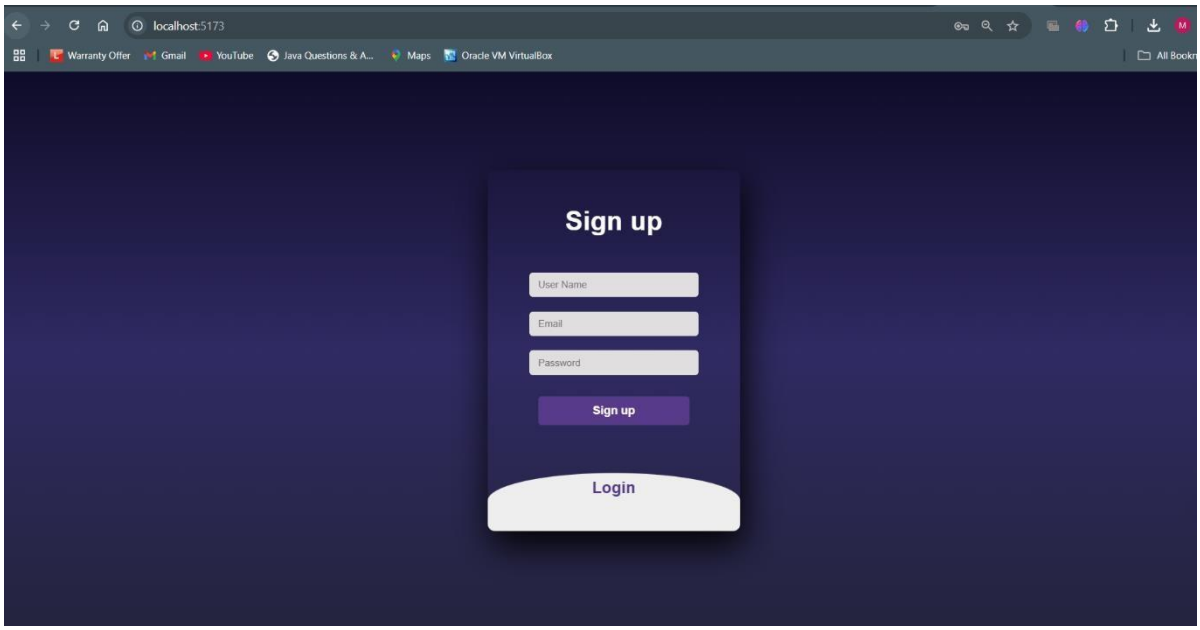


FIGURE 8.1

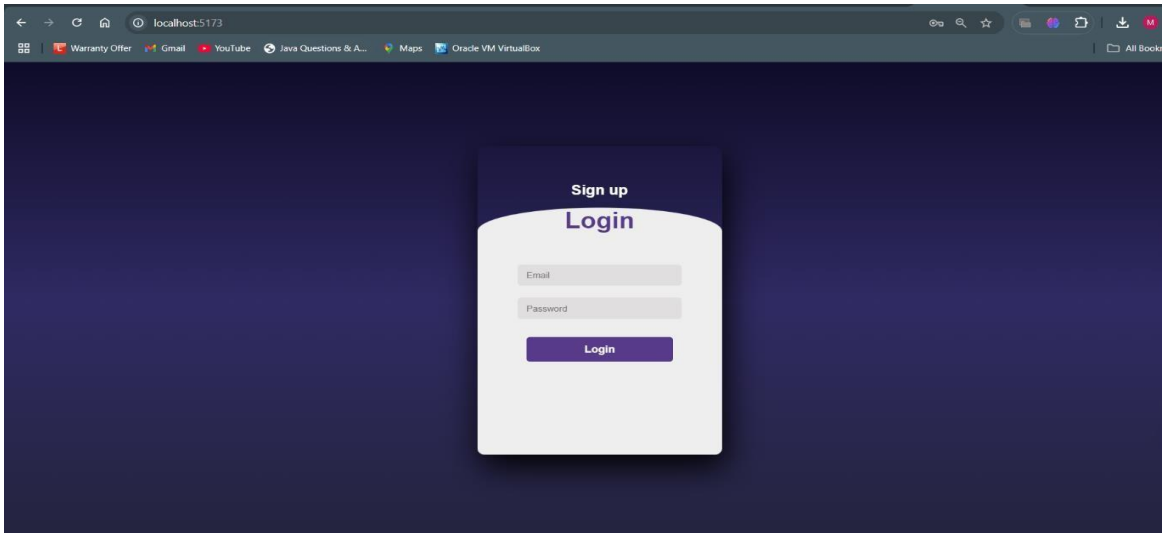


FIGURE 8.2

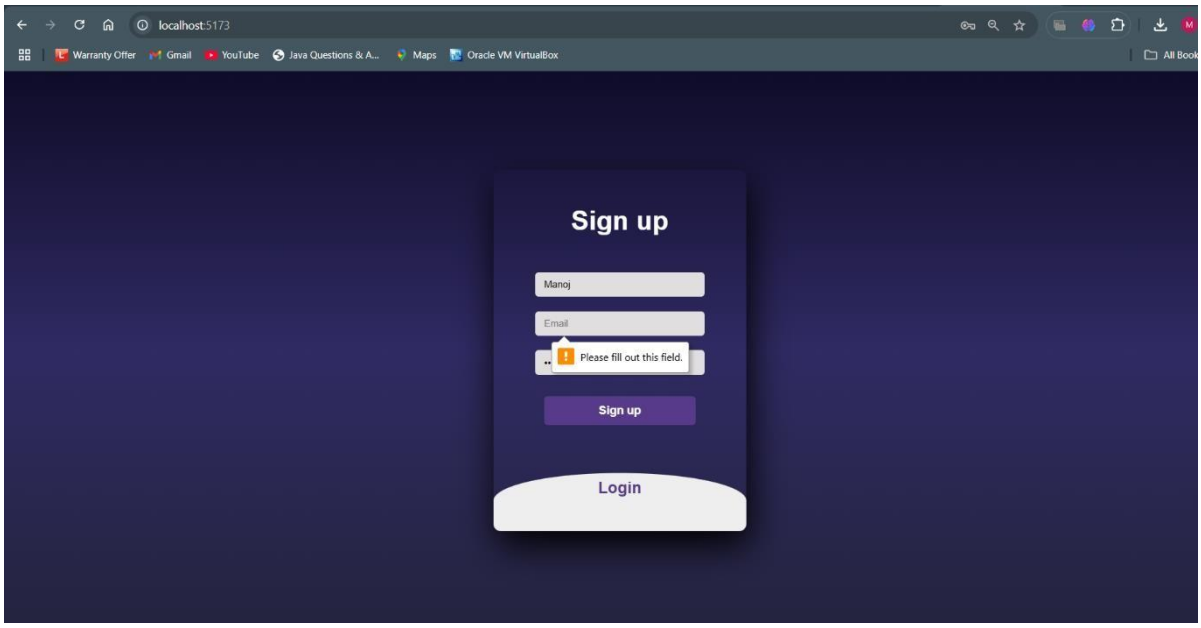


FIGURE 8.3

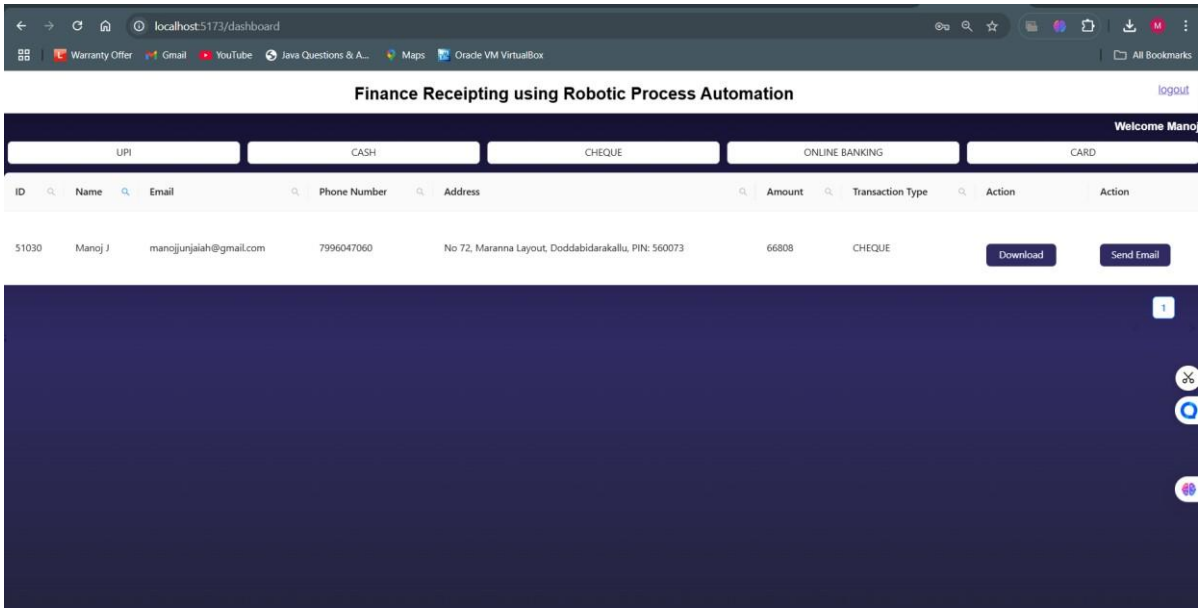


FIGURE 8.4

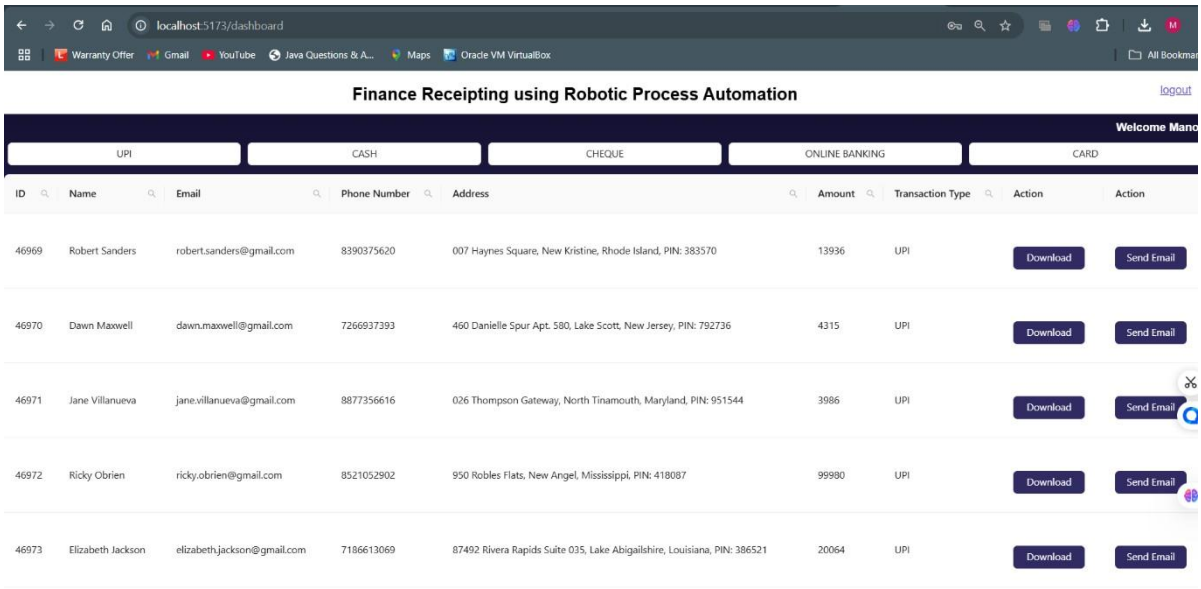


FIGURE 8.5

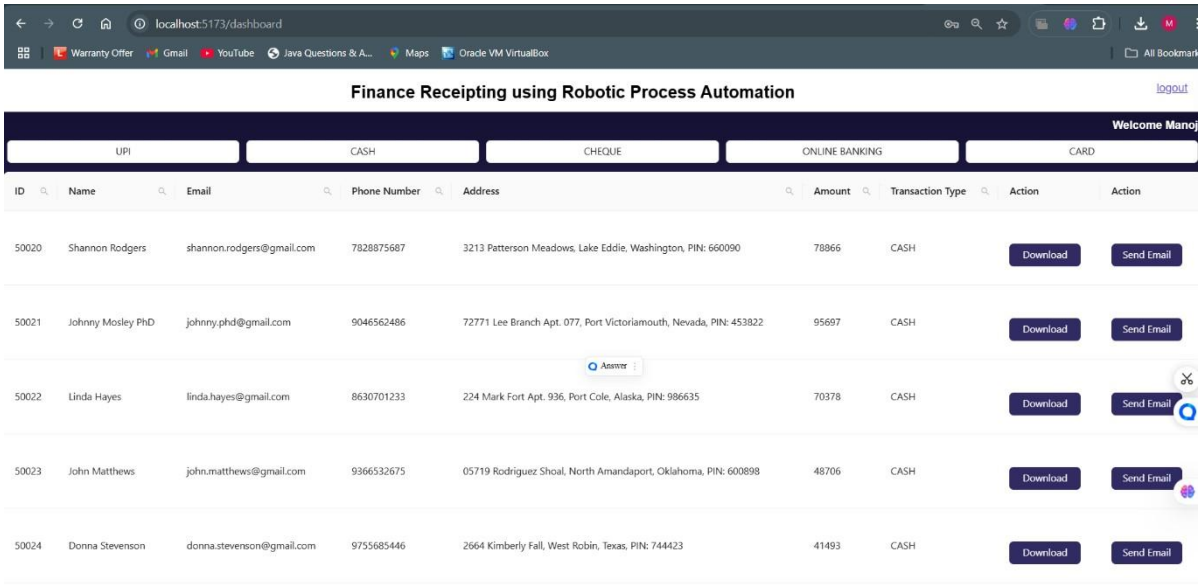


FIGURE 8.6

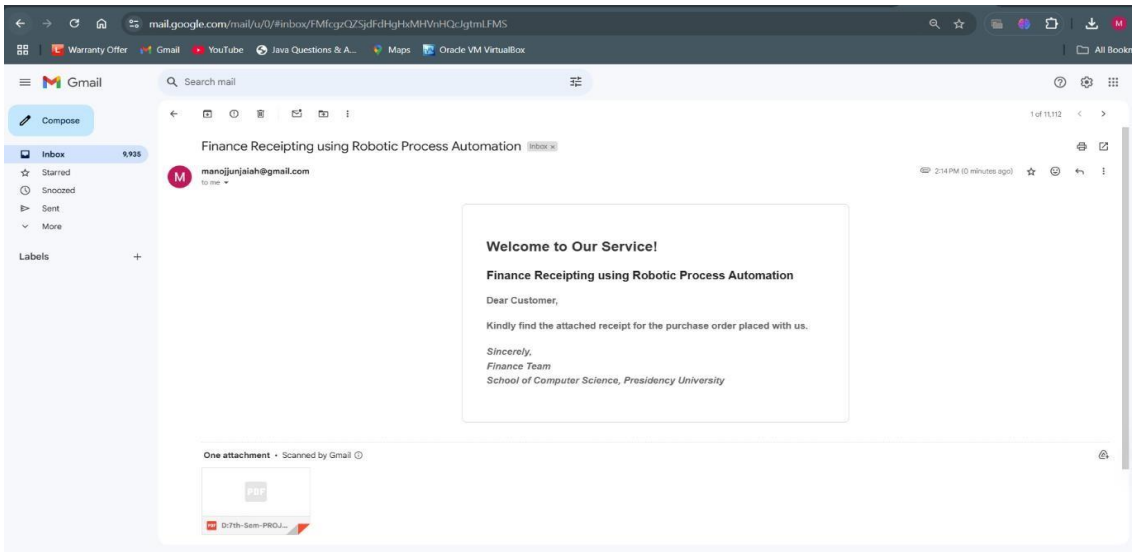


FIGURE 8.7

Transaction Receipt

Date: 2025-01-15
Receipt Number: RCPT-51030-1736930657

Customer Name	Manoj J
Customer ID	51030
Email Address	manojjunjaiah@gmail.com
Phone Number	7996047060
Address	No 72, Maranna Layout, Doddabidarakallu, PIN: 560073
Transaction Amount	Rs 66808
Transaction Type	CHEQUE
Purchase Date	24-12-2022 00:00
Product Code	P3954
Product Description	Reliable Sports Car
Product Quantity	55
Cheque Transaction ID	IAZCO33Q7HNPCDR
Bank Name	HDFC
Acount Number	5225289011
Cheque Number	258319
Ifsc	HDFC6924947 s
Date	24-12-2022 00:00

Thank you for your transaction!

Team RPA, 7th Semester
School of Computer Science, Presidency University

FIGURE 8.8

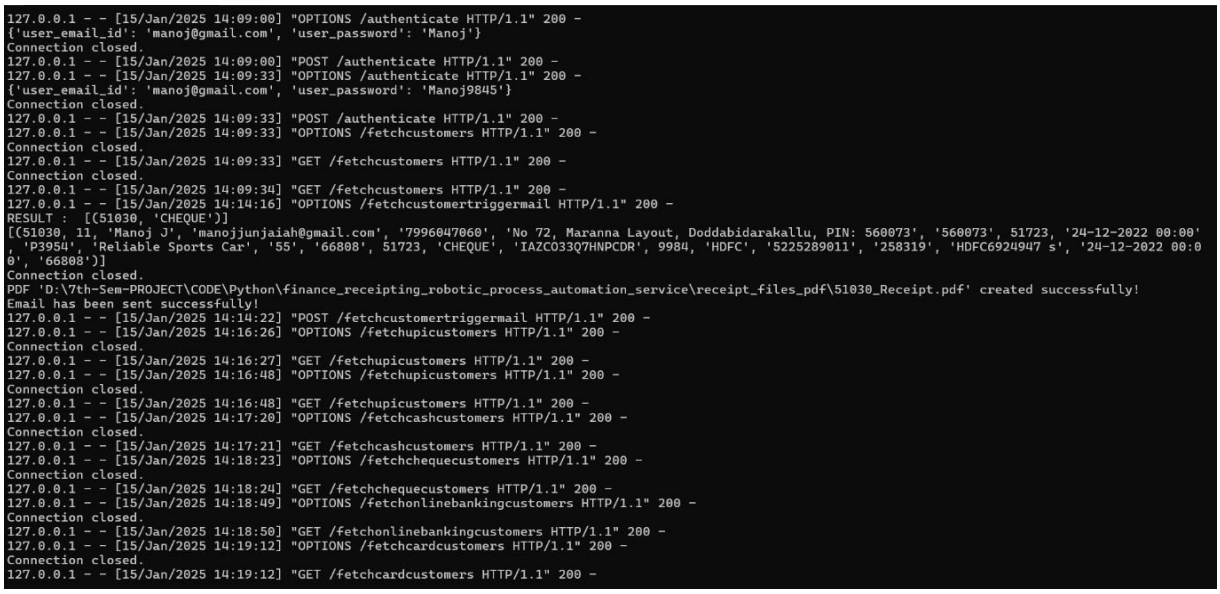


FIGURE 8.9

FIGURE 8.10

APPENDIX-C
ENCLOSURES

International Journal For Multidisciplinary Research

An International Open Access Peer Reviewed Journal • Impact Factor: 9.24 • E-ISSN: 2582-2160

Certificate of Publication

The editorial board of IJFMR is hereby awarding the certificate of publication to

Manoj J

in recognition of publication of the paper titled

Finance Receipting using Robotic Process Automation

Co-Authors: Mohammed Maaz Rehman, Anjan GM, Prasad PS

Published In: Volume 6, Issue 6 (November–December 2024)

Paper Id: 33147

www.ijfmr.com • editor@ijfmr.com

Editor / Publisher
IJFMR

International Journal For Multidisciplinary Research

An International Open Access Peer Reviewed Journal • Impact Factor: 9.24 • E-ISSN: 2582-2160

Certificate of Publication

The editorial board of IJFMR is hereby awarding the certificate of publication to

Mohammed Maaz Rehman

in recognition of publication of the paper titled

Finance Receipting using Robotic Process Automation

Co-Authors: Manoj J, Anjan GM, Prasad PS

Published In: Volume 6, Issue 6 (November–December 2024)

Paper Id: 33147

www.ijfmr.com • editor@ijfmr.com

Editor / Publisher
IJFMR

International Journal For Multidisciplinary Research

An International Open Access Peer Reviewed Journal • Impact Factor: 9.24 • E-ISSN: 2582-2160

Certificate of Publication

The editorial board of IJFMR is hereby awarding the certificate of publication to

Anjan GM

in recognition of publication of the paper titled

Finance Receipting using Robotic Process Automation

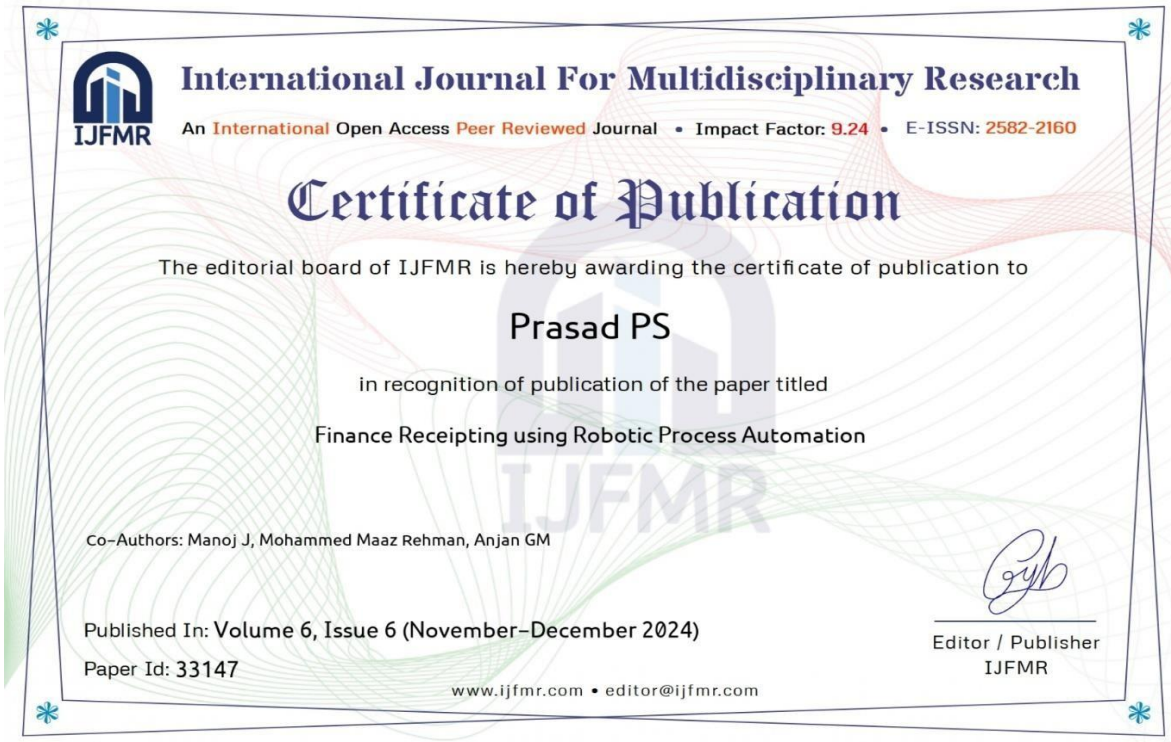
Co-Authors: Manoj J, Mohammed Maaz Rehman, Prasad PS

Published In: Volume 6, Issue 6 (November-December 2024)

Paper Id: 33147

Editor / Publisher
IJFMR

www.ijfmr.com • editor@ijfmr.com



International Journal For Multidisciplinary Research

An International Open Access Peer Reviewed Journal • Impact Factor: 9.24 • E-ISSN: 2582-2160

Certificate of Publication

The editorial board of IJFMR is hereby awarding the certificate of publication to

Prasad PS

in recognition of publication of the paper titled

Finance Receipting using Robotic Process Automation

Co-Authors: Manoj J, Mohammed Maaz Rehman, Anjan GM

Published In: Volume 6, Issue 6 (November-December 2024)

Paper Id: 33147

Editor / Publisher
IJFMR

www.ijfmr.com • editor@ijfmr.com

APPENDIX – C

ENCLOSURES

Plagiarism/Similarity Report

ORIGINALITY REPORT

14%	9%	6%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Symbiosis International University Student Paper	9%
2	Submitted to NCC Education Student Paper	1%
3	Submitted to Presidency University Student Paper	1%
4	vbook.pub Internet Source	1%
5	Submitted to M S Ramaiah University of Applied Sciences Student Paper	1%
6	www.coursehero.com Internet Source	1%
7	Submitted to University of Lancaster Student Paper	<1%
8	aimarketingmonster.com Internet Source	<1%
9	www.scrapingdog.com	

	Internet Source	<1%
10	nzmanufacturer.co.nz Internet Source	<1%
11	Submitted to University of Greenwich Student Paper	<1%
12	digitalcommons.usf.edu Internet Source	<1%
13	www.hyland.com Internet Source	<1%
14	scammo.blogspot.com Internet Source	<1%
15	fastercapital.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

2. Details of Mapping the project with Sustainable Development Goals (SDGs).

SDG 9: Industry, Innovation, and Infrastructure

Goal: Build resilient infrastructure, promote inclusive and sustainable industrialization, and foster innovation.

- Relevance to Your Project:
  - By automating financial receipting, your project introduces innovative technologies (RPA) to streamline operations, reduce errors, and improve efficiency.
  - Enhances digital infrastructure by transitioning from manual processes to automated systems.
  - Promotes sustainable practices in financial systems, ensuring robust and adaptable solutions for the future.
- Key Targets Your Project Supports:
  - 9.4: Upgrade infrastructure and industries to make them sustainable, with increased resource use efficiency and greater adoption of clean and environmentally sound technologies.
  - 9.5: Enhance scientific research and technological innovation.

2. SDG 12: Responsible Consumption and Production

Goal: Ensure sustainable consumption and production patterns.

- **Relevance to Your Project**:
  - Your project reduces reliance on **paper-based processes**, contributing to **resource conservation** and waste reduction.
  - Encourages companies to adopt **sustainable practices** by integrating RPA into their operations.
  - Supports the creation of efficient and eco-friendly financial workflows.
- **Key Targets Your Project Supports**:
  - **12.2**: Achieve sustainable management and efficient use of natural resources.
  - **12.5**: Substantially reduce waste generation through prevention, reduction, recycling, and reuse.