Solution Design Document

# Contract Intake and Validation System

### Version 1.0

### Date: [Current Date]

---

## Table of Contents

1. 1. Introduction
2. 2. System Architecture
3. 3. Component Details
4. 4. Workflow Description
5. 5. Output Structure
6. 6. Error Handling & Logging
7. 7. Integration Points
8. 8. Deployment & Dependencies
9. 9. Future Enhancements

---

## 1. Introduction

### 1.1 Purpose  
The Contract Intake and Validation System automates the end-to-end process of contract processing,   
from initial retrieval to final validation and storage. It combines Robotic Process Automation (RPA)   
for document handling with AI-powered analysis for intelligent contract processing.  
  
### 1.2 Problem Statement  
Manual contract processing faces several challenges:  
- Time-consuming document retrieval and processing  
- Inconsistent validation standards  
- Error-prone manual data extraction  
- Difficulty scaling with increased contract volume  
- Lack of standardized risk assessment  
  
### 1.3 Solution Overview  
Our system addresses these challenges through:  
- Automated contract retrieval and upload using UiPath RPA  
- AI-powered contract analysis using CrewAI  
- Standardized output generation in multiple formats  
- Consistent validation rules and risk assessment  
- Scalable and maintainable architecture

## 2. System Architecture

### 2.1 High-Level Architecture  
[Insert Architecture Diagram]  
  
The system consists of three main layers:  
  
1. \*\*RPA Layer\*\*  
 - Download Bot: Retrieves contracts from source system  
 - Upload Bot: Uploads processed results  
 - UiPathRunner: Manages bot execution  
  
2. \*\*Processing Layer (CrewAI)\*\*  
 - Contract Analyst: Data extraction  
 - Contract Validator: Validation and risk assessment  
 - Notification Specialist: Summary generation  
  
3. \*\*Output Layer\*\*  
 - Structured JSON data  
 - Validation reports  
 - Email summaries  
 - Excel reports  
  
### 2.2 Key Technologies  
- UiPath for RPA operations  
- CrewAI for intelligent processing  
- Python for core implementation  
- JSON for data storage  
- Excel for report generation

## 5. Output Structure

### 5.1 Directory Organization  
```  
output/  
├── vendor\_xyz\_contract/  
│ ├── extracted\_contract\_details.json  
│ ├── extracted\_contract\_details.xlsx  
│ ├── validation\_report.json  
│ ├── email\_summary.txt  
├── vendor\_novasecure\_contract/  
│ ├── ...  
```

## 6. Error Handling & Logging

### 6.1 RPA Error Handling  
```python  
def execute\_process(self, process\_file: str) -> bool:  
 try:  
 if not os.path.exists(self.uipath\_path):  
 logger.error(f"UiRobot.exe not found at {self.uipath\_path}")  
 return False  
  
 cmd\_str = f'"{self.uipath\_path}" execute --file "{process\_file}"'  
 logger.info(f"Executing UiPath process: {process\_file}")  
 result = subprocess.run(cmd\_str, shell=True, capture\_output=True, text=True)  
 return result.returncode == 0  
 except Exception as e:  
 logger.error(f"Error executing UiPath process: {str(e)}")  
 return False  
```

# 7. Integration Points

## 7.1 RPA Integration Components

### 7.1.1 RPA Configuration (rpa\_config.py)

# RPA Bot Configuration  
SKIP\_RPA\_BOTS = False # Set to True to skip bot execution during testing  
MAX\_RETRIES = 3 # Number of times to retry bot execution on failure  
RETRY\_DELAY = 5 # Seconds to wait between retries  
  
# Bot package paths  
DOWNLOAD\_BOT\_PATH = r"C:\Users\bened\OneDrive\Desktop\Contract\_Intake\_Automation\contract\_download\_bot\ContractDownloadBot.1.0.2.nupkg"  
UPLOAD\_BOT\_PATH = r"C:\Users\bened\OneDrive\Desktop\Contract\_Intake\_Automation\contract\_output\_upload\_bot\ContractOutputUploadBot.1.0.1.nupkg"  
  
# UiPath Configuration  
UIPATH\_CLI\_PATH = r"C:\Users\bened\AppData\Local\Programs\UiPath\Studio\UiRobot.exe"

### 7.1.2 RPA Utilities (rpa\_utils.py)

import logging  
from pathlib import Path  
  
logger = logging.getLogger(\_\_name\_\_)  
  
def download\_contracts() -> bool:  
 logger.info("Executing Contract Download Bot...")  
 return True  
  
def upload\_outputs() -> bool:  
 logger.info("Executing Contract Output Upload Bot...")  
 return True

# 8. Deployment & Dependencies

## 8.1 Package Structure

contract\_intake\_agents/  
├── src/  
│ └── contract\_intake\_agents/  
│ ├── \_\_init\_\_.py  
│ ├── main.py  
│ ├── rpa\_utils.py  
│ ├── config/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── agents.yaml  
│ │ ├── rpa\_config.py  
│ └── utils/  
│ └── uipath\_runner.py

## 8.2 Dependencies (setup.py)

from setuptools import setup, find\_packages  
  
setup(  
 name="contract\_intake\_agents",  
 version="0.1.0",  
 packages=find\_packages(where="contract\_intake\_agents/src"),  
 package\_dir={"": "contract\_intake\_agents/src"},  
 install\_requires=[  
 "crewai",  
 "crewai\_tools",  
 ],  
)

# 9. Future Enhancements

## 9.1 Planned Improvements

- Enhanced Error Recovery  
- Parallel Processing  
- Real-Time Monitoring Dashboard  
- Improved API Integration for Contract Uploads

# Appendix A: Installation Guide

1. Clone the repository.  
2. Install required dependencies.  
3. Configure RPA paths.  
4. Run the main script using: python main.py  
5. Verify outputs in the output/ folder.

# Appendix B: Troubleshooting Guide

1. Check RPA bot paths and permissions.  
2. Verify system resources and environment variables.  
3. Review log files for error messages.  
4. Restart the application if issues persist.