AutoDoc API Research Paper

Mr. Arvind Pawar
Aarti Chauhan Bhumika Kandari Anas
Rishabh Kashyap

Introduction

We live in a time where businesses are generating more documents than ever before. Among all of these, the Purchase Order stands out as one of the most important — it captures what a company is buying, how much, and from whom. Yet despite how critical these documents are, a surprising number of organizations still process them by hand. Someone sits down, opens a PDF, reads through it, and types the information into a system. It works, but it is slow, repetitive, and leaves a lot of room for mistakes.

This project was built to change that. The idea is simple: instead of having a person extract information from a Purchase Order PDF, let a system do it automatically. The PDF gets uploaded, the system reads it, pulls out the important details, and saves everything neatly into a database. What might take a person several minutes takes the system just a few seconds.

Beyond the time savings, there is a bigger picture here. As businesses grow, so does the number of documents they deal with. A team that can manage fifty POs a day cannot simply manage five hundred without either making mistakes or burning out. Automation solves that problem. This project is ultimately about making business operations more sustainable, more accurate, and more ready for growth.

Problem Statement

The challenge with Purchase Orders is that no two of them look exactly the same. Every supplier has their own template, their own layout, their own way of organizing information. One PO might list the product name at the top of a table; another might bury it halfway down the page. When a human reads through these, they can adapt on the fly. A computer, however, needs clear instructions
— and with so much variation, those instructions are hard to write.

This variation leads to a real problem at scale. When the volume of incoming POs is small, manual entry is manageable. But as volume grows, so does the workload. Staff end up spending hours on data entry that adds no creative or strategic value. Mistakes creep in — a wrong quantity, a misread barcode, a product name entered with a typo. These errors might seem small, but they can cause serious issues down the line: wrong items get ordered, invoices don't match, suppliers get frustrated.

What this project sets out to address is that gap between what the documents contain and what the system needs. It asks a practical question: how do we design something that can reliably read through a PDF Purchase Order, understand what it is looking at, and pull out the right information without needing a human to guide it every step of the way?

Objectives

The goal of this project is to build a working, practical system that automates the extraction of data from Purchase Order PDFs. At its core, the system needs to do three things well: read the PDF, find the relevant information, and store it properly. The key fields it targets are product names, quantities, and barcodes — the three pieces of information that most procurement workflows depend on.

Accuracy matters just as much as speed. There is no point in processing documents quickly if the output is unreliable. So the system is designed to extract data with a high degree of confidence, and to flag anything it is unsure about rather than guessing. This way, the small percentage of documents that the system cannot handle cleanly can be reviewed by a person, while the vast majority get processed without any human involvement.

Scalability is also built into the objective. The system should not struggle whether it is processing ten documents or ten thousand. The use of a proper relational database ensures that data is stored in an organized, retrievable way — not just dumped into a file somewhere. The end result should be something that a real business could actually use, not just a prototype that works in ideal conditions.

Literature Review

There is no shortage of tools that can extract text from a PDF. Basic utilities like PDF2Text have existed for years, and software like Adobe Acrobat offers export features that can convert a PDF into plain text or a spreadsheet. The problem with most of these tools is that they treat the document as a stream of characters rather than as a structured object. The result is often a jumbled mess of text that has lost all sense of layout and meaning.

More sophisticated approaches do exist. Layout-aware extraction tools try to understand the structure of a document by identifying text blocks and their positions on the page. Machine learning techniques, particularly Named Entity Recognition, go further by attempting to understand what the text actually means
— labeling words as product names, dates, quantities, and so on. These methods are impressive, but they come with significant costs: they require large amounts of labeled training data, substantial computational resources, and a level of technical expertise that is not always available.

For a system designed to handle structured business documents like Purchase Orders, these heavyweight approaches are often overkill. POs follow patterns. They have consistent labels like 'Product', 'Quantity', and 'Barcode'. A well-crafted set of rules can reliably find these patterns without needing a neural network. This project takes that simpler, more transparent approach — one that is easier to build, easier to understand, and easier to maintain.

System Architecture

The system is organized into four distinct layers, each with a clear responsibility. This was a deliberate design choice. When everything is tangled together in a single block of code, a change in one part can unexpectedly break something else. By separating concerns cleanly, the system becomes much easier to work with — and much easier to improve over time.

The first layer handles ingestion. This is where the user interacts with the system, uploading a PDF through a simple web interface. The second layer is responsible for extraction — it takes the uploaded file and uses a PDF parsing library to pull out the raw text. The third layer does the actual work of finding meaning in that text. It applies pattern-matching rules to identify the fields the system cares about and cleans up whatever it finds. Finally, the fourth layer takes the cleaned, structured data and saves it into a relational database.

Each layer can be upgraded independently. If a better PDF parsing library comes along, only the extraction layer needs to change. If the business needs to start capturing a new field, only the processing layer gets updated. This kind of modular thinking is what separates a project that stays useful for years from one that becomes a maintenance burden almost immediately.

Methodology

The process of extracting data from a Purchase Order PDF follows a clear sequence of steps. First, the document is opened and its text content is read out in full. This raw text is essentially a long string of characters — useful, but not yet organized in any meaningful way. The next step is where the real work begins.

Regular expressions are used to search through the raw text and find the data the system is looking for. These are patterns written in a specific syntax that can match things like 'a label followed by a colon, followed by a value'. For example, a pattern designed to find the quantity might look for the word 'Qty' or 'Quantity', followed by a number. When the pattern finds a match, that matched value gets extracted and passed along to the next stage.

Before anything gets stored in the database, the extracted values go through a cleaning process. Whitespace gets trimmed, numbers get normalized so that '1,000' and '1000' are treated the same way, and barcodes get checked against their expected format. If something looks wrong — a barcode that is the wrong length, a quantity that is empty — it gets flagged for review rather than silently inserted with bad data. This careful, step-by-step approach is what gives the system its reliability.

Implementation

Python was the natural choice for building this system. It has a rich ecosystem of libraries for exactly this kind of work, and its readability makes the code easier to review and maintain. Flask handles the web interface — it is a lightweight framework that is perfect for a focused application like this, without the overhead of a larger platform. PyPDF2 takes care of reading the PDF files, and the built-in regular expression library handles the pattern matching.

For storage, MySQL was selected because the data being handled is genuinely relational. A Purchase Order has a header — a PO number, a date, a supplier — and it has line items, each of which belongs to that header. A relational database captures these relationships naturally and makes it easy to query and report on the data later. Storing everything in a flat file or a NoSQL database would have made that harder.

Error handling was not an afterthought. The system checks that uploaded files are actually PDFs before trying to process them. If the text extraction comes back empty, it logs the event and moves on rather than crashing. If a regex pattern finds no match, the field is stored as null with a flag indicating that human review is needed. If a database insert fails for any reason, the entire record is rolled back so there are no half-saved, inconsistent entries. These protections make the system trustworthy in practice, not just in theory.

Results

When tested against a range of Purchase Order documents, the system performed well. It successfully identified and extracted product names, quantities, and barcodes across multiple different document layouts, storing each record correctly in the database. Compared to manually entering the same data by hand, the time savings were significant — what might take a person several minutes per document was completed by the system in a matter of seconds.

Consistency was another notable result. Human data entry tends to vary depending on how tired or focused the person is at a given moment. The system, by contrast, applies the same rules every single time. As long as the document follows a recognizable pattern, the output is predictable and uniform. This consistency is particularly valuable when the extracted data feeds into downstream systems like inventory management or invoicing.

The testing also revealed the system's boundaries. Documents with unusual layouts or those that deviated significantly from the patterns the system was trained on produced incomplete results. These cases were correctly flagged rather than silently producing wrong data, which is the right behavior — but it does highlight that the system works best when the input documents are reasonably consistent in structure.

Advantages

The most immediate advantage of this system is time. Processing a Purchase Order manually takes anywhere from three to eight minutes depending on its complexity. The automated system handles the same task in under five seconds. At any meaningful volume, that difference compounds into hours of recovered staff time every single week — time that can be redirected toward work that actually requires human judgment.

Accuracy improves alongside speed. Manual data entry carries an inherent error rate, typically somewhere between one and five percent. A single mistyped barcode or wrong quantity might seem trivial, but in a procurement context it can trigger the wrong order, create an inventory mismatch, or cause a dispute with a supplier. Rule-based extraction, applied consistently, eliminates the cognitive fatigue and distraction that drive those errors.

There is also a longer-term strategic advantage. By building on an open-source stack with a clean, modular architecture, the organization avoids vendor lock-in and keeps full control over its data. The system creates a structured, queryable database from what were previously unstructured PDFs — and that data asset has value beyond just processing. It enables trend analysis, supplier performance tracking, and the kind of reporting that supports better business decisions.

Limitations

The most significant limitation of the current system is its dependence on document format. The regular expressions that drive the extraction are written against specific patterns — and if a supplier changes their PO template, those patterns may no longer match. This means the system requires ongoing maintenance. Each time a new supplier is onboarded or an existing one changes their format, someone needs to review and potentially update the extraction rules.

Scanned documents present a harder problem. When a PDF is created by scanning a physical document, there is no underlying text layer for the system to read — only an image. PyPDF2 cannot extract anything useful from an image-only PDF, which means a large category of real-world documents is simply out of scope for the current version. Handling these would require an OCR step to first convert the image into text before any extraction can take place.

Complex layouts also cause difficulties. If a document uses multiple columns, embedded images, or data presented in intricate tables, the text extraction may come out in a confused order that breaks the regex patterns. These are not edge cases in the real world — many professional PO templates use exactly these kinds of designs. Acknowledging these limitations honestly is important, because it sets realistic expectations and points directly toward where future development effort should go.

Future Scope

The most logical next step for this system is adding support for scanned documents through Optical Character Recognition. Libraries like Tesseract can convert a scanned PDF page into readable text, which would then feed directly into the existing extraction pipeline. This single addition would dramatically expand the range of documents the system can handle, bringing in the large portion of real-world POs that exist only as physical scans.

Beyond OCR, there is a compelling case for introducing machine learning at the pattern recognition stage. Instead of writing regex rules by hand for each new supplier format, a trained model could learn to identify fields from examples. This would reduce maintenance overhead and make the system far more adaptable. Newer approaches — including large language models — can extract structured information from unstructured text with very little setup, and integrating one as a fallback for documents that fail regex extraction could significantly improve overall coverage.

Further down the road, connecting the system to enterprise platforms like ERP or inventory management software would make it part of a fully automated procurement workflow. A Purchase Order could arrive by email, get automatically processed, and have its data inserted into the business's systems without any human involvement from start to finish. That is the vision this project is working toward — and the foundation it builds is a solid step in that direction.

Conclusion

What this project demonstrates, above all, is that automation does not need to be complicated to be valuable. By combining straightforward tools — a PDF reader, regular expressions, a web interface, and a database — it is possible to build something that saves real time, reduces real errors, and scales in a way that manual processes simply cannot.

The system has limitations, and this paper has been honest about them. It works best on consistently formatted text-based PDFs, and it will need ongoing attention as supplier formats change. But these are manageable challenges, not fundamental flaws. The modular architecture was designed precisely to make future improvements easy — each limitation points toward a concrete next step rather than a dead end.

Perhaps the most important thing this project shows is that the jump from manual processing to intelligent automation does not require a massive investment in cutting-edge AI. A well-designed, rule-based system built on open-source tools can deliver most of the value at a fraction of the cost and complexity. That is a lesson worth carrying into any future project: start with what works, build it well, and leave the door open for what comes next.