Project Overview

I developed an OCR-driven inventory management system to process purchase orders in a specialized PDF format. The system extracts structured item-level data — Quantity, Catalog #, Description, Unit Price, Total, and Date Ordered — while also generating a default Date Received (set to 7 days later by algorithm, user-editable). All extracted fields are presented in a Django-based UI where users can validate, edit, and export results.

The project originally included barcode scanning through InvenTree, based on an assumption that it would improve tracking. After testing the workflow, my professor realized scanning added unnecessary steps and was inconvenient. I then repurposed the code into a standalone Django app focused purely on OCR extraction and editable UI.

System Workflow

1. **PDF Upload** → User uploads a purchase order; unreadable files are flattened and OCR'd.
2. **Extraction** → Regex + positional parser captures item fields:
    - Quantity
    - Catalog #
    - Description
    - Unit Price
    - Total
    - Date Ordered (from PO)
    - Date Received (algorithm default +7 days, user-editable)
3. **Django UI** → Results shown in an **editable table** with features:
    - Inline editing of all fields
    - Calendar date pickers for Date Ordered/Received
    - Row add/remove controls for corrections
    - Validation before export
4. **Export/Reporting** → Clean structured output for records.

Challenges Encountered

1. **Feature Assumptions**
    - Barcode scanning was assumed to improve traceability.
    - **Reality**: Workflow testing showed scanning slowed the process.
    - **Solution**: Pivoted to a standalone extraction + UI system.
2. **Header Variability**
    - Could not build universal parsing.
    - **Outcome**: Specialized extractor for one PO layout.
3. **OCR Quality**
    - Poor scans lacked embedded text.
    - **Solution**: Added Ghostscript + OCRmyPDF preprocessing.
4. **User Usability**
    - OCR output sometimes had small errors.
    - **Solution**: Editable Django UI with validation and correction tools.

My Contributions

- **Full Stack Development**: Built end-to-end Django app (backend OCR + frontend UI).
- **Extraction Pipeline**: Regex + positional parser for all item-level fields.
- **Date Algorithm**: Auto-generated Date Received (+7 days), editable in UI.
- **UI Features**: Editable tables, date pickers, row editing, export.
- **Pivot & Refactor**: Transitioned from InvenTree + barcodes to standalone app after workflow testing.

Outcome and Lessons Learned

The system automated purchase order parsing and gave users full control to validate/correct results, making it more practical than the original barcode-heavy design. A key lesson was that real workflows sometimes invalidate assumptions — testing revealed scanning was more inconvenient than helpful.

What I Learned

- The importance of **prototyping and testing** before committing to features.
- How to design **semi-automated workflows** with editable user interfaces.
- That solving the *real problem* matters more than over-engineering features.

What I Would Do Differently

- Confirm assumptions with **workflow testing early** in development.
- Document expectations before starting to avoid wasted effort.
- Add support for multiple PO layouts and a persistent database backend.