ERPNext OCR Learning App - Complete Technical Specification

1. Project Overview

1.1 Purpose

An intelligent OCR-based invoice processing application for ERPNext that learns from historical data and user corrections to automate Purchase Invoice and Journal Voucher creation with high accuracy.

1.2 Key Innovation

Learning-based system that remembers supplier-specific patterns for:

- Item mapping variations
- UOM conversions
- Tax/HSN code assignments
- Payment terms interpretation

1.3 Market Gap Analysis

Existing Solutions:

- Invoice OCR (Frappe Cloud) - Basic OCR without learning
- Invoice2ERPNext - External service with basic mapping, no learning
- Monogramm/erpnext_ocr - Basic overlay mapping

Critical Missing Features:

- ✗ No supplier-specific learning
- ✗ No UOM conversion intelligence
- ✗ No tax/HSN learning by context
- ✗ No historical pattern recognition

Our Competitive Advantage:

- ✅ 80% automation through learning
- ✅ 3 years historical data training
- ✅ Intelligent UOM conversion
- ✅ Context-aware supplier patterns

---

2. Technical Architecture

2.1 Implementation Approach

Custom Frappe App - NOT core doctype modification

- App Name: `invoice_ocr`
- Update-safe and modular design
- Clean integration with ERPNext core

2.2 Integration Strategy

Option A: Custom Buttons (Recommended)

```javascript
// Add buttons to PI/JV list views
frappe.ui.form.on('Purchase Invoice', {
    refresh: function(frm) {
        frm.add_custom_button('OCR Upload', () => {
            // Opens OCR processing dialog
        });
    }
});
```

Option B: Standalone Custom Page

- Page: "OCR Invoice Processor"
- User selects document type during processing

---

3. Data Structure & DocTypes

3.1 Primary DocType: "Invoice OCR Processor"

Naming Series: OCRP.YYYY.MM.##### (OCRP2526001, OCRP2526002...)

Purpose: Permanent audit trail + learning source

Key Fields:

```python
```

```
{
  # Basic Info
  "supplier": "Link to Supplier",
  "invoice_number": "Data (from OCR)",
  "invoice_date": "Date (from OCR)",
  "payment_terms": "Select (Cash/Credit/15 Days/30 Days etc.)",

  # OCR Processing
  "extracted_items": "Table Field",
  "final_mappings": "Table Field",
  "ocr_status": "Select (Draft/Processing/Ready/Completed)",

  # Learning & Audit
  "created_document": "Dynamic Link (PI/JV)",
  "total_amount": "Currency",
  "learning_confidence": "Percent",
  "user_corrections": "Long Text",

  # File Management
  "invoice_attachment_status": "Select (Attached to Created Doc)"
}
```

Child Table: Extracted Items

python

```
{
  "ocr_item_text": "Data",
  "suggested_erpnext_item": "Link to Item",
  "final_erpnext_item": "Link to Item",
  "ocr_quantity": "Float",
  "ocr_uom": "Data",
  "ocr_rate": "Currency",
  "ocr_amount": "Currency",

  # Conversion Logic
  "erpnext_quantity": "Float",
  "erpnext_uom": "Link to UOM",
  "erpnext_rate": "Currency",
  "conversion_factor": "Float",
  "conversion_notes": "Text"
}
```

3.2 Learning Database: "Supplier Item Mapping"

Purpose: Permanent learning storage for pattern recognition

Fields:

```python
{
    "supplier": "Link to Supplier",
    "ocr_item_text": "Data",
    "erpnext_item_code": "Link to Item",
    "uom_conversion_pattern": "JSON",
    "tax_template": "Link to Item Tax Template",
    "hsn_code": "Data",
    "payment_terms_pattern": "Data",

    # Learning Metrics
    "frequency_count": "Int",
    "confidence_score": "Percent",
    "last_used": "Datetime",
    "user_correction_count": "Int",
    "success_rate": "Percent"
}
```

3.3 Configuration: "OCR Settings"

Fields:

```python
{
    "ocr_engine": "Select (Tesseract/Google Vision API)",
    "api_credentials": "Password",
    "confidence_threshold": "Percent",
    "auto_submit_threshold": "Percent",
    "learning_enabled": "Check",
    "historical_data_processed": "Check"
}
```

---

4. Core Learning Logic

4.1 Comprehensive Learning Matrix

All Parameters Learned:

```python
```

```
learning_record = {
  "supplier": "ABC Motors",
  "invoice_patterns": {
    "item_mapping": {
      "Grease 2kg - 1 Pcs": {
        "erpnext_item": "Grease",
        "conversion": {
          "supplier_qty": 1,
          "supplier_uom": "Pcs",
          "supplier_spec": "2kg per piece",
          "erpnext_qty": 2,
          "erpnext_uom": "Kg",
          "rate_per_erpnext_uom": 100
        }
      }
    },
    "tax_patterns": {
      "Wiper Blade": {
        "context": "automotive_supplier",
        "hsn_code": "8512",
        "tax_template": "Auto Parts - 18%"
      }
    },
    "payment_terms": {
      "15 Days Credit": "15 Days",
      "Credit": "30 Days",
      "Net 15": "15 Days"
    }
  },
  "confidence_metrics": {
    "overall_accuracy": 95,
    "item_mapping_confidence": 98,
    "uom_conversion_confidence": 92,
    "tax_assignment_confidence": 89
  }
}
```

4.2 UOM Conversion Intelligence

Core Logic:

```
python
```

```
def convert_supplier_to_erpnext_uom(ocr_data, erpnext_item):
    # Get ERPNext item's stock UOM (NEVER change this)
    erpnext_uom = get_item_stock_uom(erpnext_item)

    # Parse supplier specification
    supplier_spec = parse_item_specification(ocr_data.item_text)

    # Example: "Grease 2kg - 1 Pcs - INR 200"
    conversion = {
        "supplier_quantity": 1,
        "supplier_uom": "Pcs",
        "supplier_spec": "2kg per piece",
        "erpnext_quantity": 2,  # 1 piece × 2kg
        "erpnext_uom": "Kg",
        "erpnext_rate": 100,    # 200 ÷ 2kg = 100/kg
        "total_verification": True  # 2 × 100 = 200 ✓
    }
    return conversion
```

4.3 Historical Data Migration Strategy

Phase 1: Data Import (3 Years)

python

```
def migrate_historical_data():
    # Scan existing Purchase Invoices (3 years)
    historical_invoices = get_purchase_invoices(
        from_date="2022-01-01",
        to_date="2025-01-01"
    )

    # Build initial learning patterns
    for invoice in historical_invoices:
        extract_learning_patterns({
            "supplier": invoice.supplier,
            "items": invoice.items,
            "payment_terms": invoice.payment_terms,
            "project": invoice.project,
            "taxes": invoice.taxes
        })

    # Create confidence baseline
    build_initial_confidence_matrix()
```

---

5. Dynamic Mandatory Fields System

5.1 Dynamic Field Detection

Auto-Discovery of Mandatory Fields:

```python
def get_dynamic_mandatory_fields(doctype):
    """Dynamically fetch mandatory fields for any doctype"""
    meta = frappe.get_meta(doctype)
    mandatory_fields = []

    for field in meta.fields:
        if field.reqd == 1:  # Required field
            mandatory_fields.append({
                "fieldname": field.fieldname,
                "fieldtype": field.fieldtype,
                "label": field.label,
                "options": field.options if field.fieldtype in ["Link", "Select"] else None
            })

    return mandatory_fields

def handle_missing_mandatory_fields(doctype, data):
    """Ask user for missing mandatory fields and learn patterns"""
    mandatory_fields = get_dynamic_mandatory_fields(doctype)
    missing_fields = []

    for field in mandatory_fields:
        if not data.get(field.fieldname):
            missing_fields.append(field)

    if missing_fields:
        # Show user interface to fill missing fields
        # Learn the pattern for future automation
        return prompt_user_for_fields(missing_fields)
```

5.2 Resilient Field Management

Error-Proof Design:

```python
```

```python
def process_document_creation(doctype, data):
    try:
        # Get current mandatory fields
        mandatory_fields = get_dynamic_mandatory_fields(doctype)

        # Fill from learned patterns
        auto_filled_data = apply_learned_patterns(data, mandatory_fields)

        # Check for missing fields
        missing = validate_mandatory_fields(auto_filled_data, mandatory_fields)

        if missing:
            # Ask user and learn
            user_input = get_user_input_for_missing_fields(missing)
            save_field_learning_pattern(data.supplier, user_input)
            auto_filled_data.update(user_input)

        # Create document
        return create_erpnext_document(doctype, auto_filled_data)

    except Exception as e:
        # Log error but continue processing
        log_error(f"Field processing error: {e}")
        return create_document_with_available_data(doctype, data)
```

---

6. Complete Purchase Cycle Automation

6.1 Document Type Selection & Processing

Primary Interface:

python

```
document_types = {
  "purchase_invoice": {
    "title": "Purchase Invoice",
    "mandatory_context": ["supplier", "items", "taxes"],
    "optional_context": ["project", "cost_center", "payment_terms"],
    "post_processing": ["payment_entry_creation", "asset_creation"]
  },
  "journal_entry": {
    "title": "Journal Voucher",
    "mandatory_context": ["accounts", "mode_of_payment"],
    "optional_context": ["project", "cost_center", "party"],
    "post_processing": ["payment_entry_linking"]
  },
  "purchase_receipt": {
    "title": "Purchase Receipt (for Assets)",
    "mandatory_context": ["supplier", "items", "warehouse"],
    "optional_context": ["project", "asset_category"],
    "post_processing": ["asset_creation", "purchase_invoice_creation"]
  }
}
```

6.2 Journal Entry Intelligence

JV Learning Patterns:

python

```
jv_learning = {
  "supplier": "ABC Motors",
  "journal_patterns": {
    "account_heads": {
      "fuel_expense": "Fuel and Transportation - Company",
      "maintenance": "Repairs and Maintenance - Company",
      "office_supplies": "Office Maintenance Expenses - Company"
    },
    "mode_of_payment": {
      "cash_payment": "Cash - Company",
      "bank_transfer": "HDFC Bank - Company",
      "credit_card": "Corporate Credit Card - Company"
    },
    "project_mapping": {
      "previous_projects": ["Project Alpha", "Project Beta"],
      "default_project": "Project Alpha",
      "confidence": 85
    }
  }
}
```

6.3 Asset Creation Workflow

Asset Processing Logic:

python

```python
def handle_asset_creation(invoice_data):
    # Detect if items are assets
    asset_items = detect_asset_items(invoice_data.items)

    if asset_items:
        # Create Purchase Receipt first
        pr = create_purchase_receipt({
            "supplier": invoice_data.supplier,
            "items": asset_items,
            "warehouse": get_learned_warehouse(invoice_data.supplier),
            "project": get_learned_project(invoice_data.supplier)
        })

        # Create Assets
        for item in asset_items:
            create_asset({
                "item_code": item.item_code,
                "purchase_receipt": pr.name,
                "asset_category": get_learned_asset_category(item.item_code),
                "project": pr.project
            })

        # Create Purchase Invoice linked to PR
        pi = create_purchase_invoice_from_receipt(pr)

        return {
            "purchase_receipt": pr.name,
            "assets_created": len(asset_items),
            "purchase_invoice": pi.name
        }
```

6.4 Project Learning & Automation

Project Intelligence:

python

```
  project_learning = {
    "supplier": "ABC Motors",
    "project_patterns": {
      "historical_projects": [
        {"project": "Office Renovation", "frequency": 15, "last_used": "2024-12-15"},
        {"project": "Vehicle Maintenance", "frequency": 8, "last_used": "2024-11-20"}
      ],
      "item_based_projects": {
        "Grease": "Vehicle Maintenance",
        "Office Supplies": "Office Renovation",
        "Electrical Items": "Office Renovation"
      },
      "auto_suggestion_confidence": 92
    }
  }

  def suggest_project(supplier, items):
    """Intelligent project suggestion based on supplier + items"""
    patterns = get_project_learning(supplier)

    # Item-based suggestion
    for item in items:
      suggested_project = patterns.item_based_projects.get(item.item_code)
      if suggested_project:
        return {
          "project": suggested_project,
          "confidence": 95,
          "reason": f"Based on item: {item.item_code}"
        }

    # Supplier-based suggestion
    most_frequent = max(patterns.historical_projects, key=lambda x: x.frequency)
    return {
      "project": most_frequent.project,
      "confidence": 80,
      "reason": f"Most frequent for supplier: {supplier}"
    }
```

---

7. Payment Entry Integration

7.1 Complete Payment Cycle

Payment Learning & Automation:

python

```
payment_patterns = {
  "supplier": "ABC Motors",
  "payment_behavior": {
    "preferred_mode": "Bank Transfer",
    "bank_account": "HDFC Bank - Company",
    "payment_timing": "15 Days Credit",
    "advance_payment_history": False,
    "partial_payment_pattern": False
  },
  "automated_payment_entry": {
    "auto_create": True,
    "confidence_threshold": 90,
    "user_approval_required": False
  }
}

def create_automated_payment_entry(purchase_invoice):
  """Create payment entry based on learned patterns"""
  supplier = purchase_invoice.supplier
  payment_pattern = get_payment_learning(supplier)

  if payment_pattern.confidence > 90:
    # Auto-create payment entry
    payment_entry = create_payment_entry({
      "payment_type": "Pay",
      "party_type": "Supplier",
      "party": supplier,
      "mode_of_payment": payment_pattern.preferred_mode,
      "paid_from": payment_pattern.bank_account,
      "paid_amount": purchase_invoice.grand_total,
      "reference_no": purchase_invoice.name,
      "reference_date": purchase_invoice.posting_date,
      "project": purchase_invoice.project
    })

    return payment_entry
  else:
    # Create draft for user review
    return create_draft_payment_entry(purchase_invoice)
```

---

8. Learning Milestones & Automation Levels

8.1 Progressive Learning System

Automation Stages:

```python
automation_levels = {
    "stage_1": {
        "invoices_processed": "1-25",
        "automation_rate": "40%",
        "user_intervention": "High",
        "focus": "Basic item mapping, UOM learning"
    },
    "stage_2": {
        "invoices_processed": "26-50",
        "automation_rate": "60%",
        "user_intervention": "Medium",
        "focus": "Tax patterns, payment terms learning"
    },
    "stage_3": {
        "invoices_processed": "51-100",
        "automation_rate": "80%",
        "user_intervention": "Low",
        "focus": "Project mapping, account head learning"
    },
    "stage_4": {
        "invoices_processed": "100+",
        "automation_rate": "95%",
        "user_intervention": "Minimal",
        "focus": "Full automation with edge case handling"
    }
}
```

8.2 Confidence-Based Processing

Smart Decision Making:

```python
```

```python
def process_invoice_with_confidence(ocr_data):
    confidence_scores = calculate_confidence(ocr_data)

    if confidence_scores.overall >= 95:
        # Full automation - create and submit
        return auto_process_and_submit(ocr_data)

    elif confidence_scores.overall >= 80:
        # Create draft for user review
        return create_draft_with_suggestions(ocr_data)

    elif confidence_scores.overall >= 60:
        # Partial automation with user input required
        return partial_automation_with_prompts(ocr_data)

    else:
        # Manual processing with OCR assistance
        return manual_processing_mode(ocr_data)
```

---

9. Enhanced DocType Structure

9.1 Updated Primary DocType Fields

"Invoice OCR Processor" - Enhanced Structure:

python

```
enhanced_fields = {
    # Document Type Selection
    "document_type": "Select (Purchase Invoice/Journal Entry/Purchase Receipt)",
    "asset_creation_required": "Check",

    # Dynamic Mandatory Fields Management
    "mandatory_fields_data": "JSON",
    "missing_fields_status": "Select (Complete/Pending User Input/Error)",
    "field_learning_applied": "Check",

    # Complete Purchase Cycle
    "purchase_receipt_created": "Link to Purchase Receipt",
    "assets_created": "Table (Link to Asset)",
    "payment_entry_created": "Link to Payment Entry",
    "journal_entry_created": "Link to Journal Entry",

    # Project & Cost Center Learning
    "suggested_project": "Link to Project",
    "project_confidence": "Percent",
    "suggested_cost_center": "Link to Cost Center",
    "cost_center_confidence": "Percent",

    # Advanced Learning Metrics
    "processing_stage": "Select (Stage 1/Stage 2/Stage 3/Stage 4)",
    "automation_percentage": "Percent",
    "user_interventions_count": "Int",
    "learning_improvements": "Long Text"
}
```

9.2 Journal Entry Learning Table

Child Table: "JV Account Learning"

python

```
jv_account_fields = {
    "account_head": "Link to Account",
    "debit_amount": "Currency",
    "credit_amount": "Currency",
    "mode_of_payment": "Link to Mode of Payment",
    "project": "Link to Project",
    "cost_center": "Link to Cost Center",
    "learning_confidence": "Percent",
    "usage_frequency": "Int"
}
```

10. Implementation Timeline & Phases

10.1 Development Phases

Phase 1: Foundation (Week 1-2)

- Basic Frappe app structure
- OCR integration (Tesseract/Google Vision)
- Core DocTypes creation
- File upload and processing

Phase 2: Learning Engine (Week 3-4)

- Historical data migration (3 years)
- Basic pattern recognition
- UOM conversion logic
- Supplier-item mapping

Phase 3: Advanced Features (Week 5-6)

- Dynamic mandatory field handling
- Complete purchase cycle integration
- Project and cost center learning
- Payment entry automation

Phase 4: Intelligence & Polish (Week 7-8)

- Confidence-based processing
- Advanced learning algorithms
- UI/UX refinements
- Testing and optimization

10.2 Success Metrics

Target Goals:

- 95% automation after 100 invoices
- 3-second average processing time
- 99% total amount accuracy
- Zero errors in UOM conversion
- 90% correct project suggestions

11. Technical Stack & Dependencies

11.1 Core Technologies

- **Framework:** Frappe Framework
- **OCR Engine:** Tesseract.js / Google Vision API
- **Language:** Python (Backend), JavaScript (Frontend)
- **Database:** MariaDB (Frappe default)

11.2 External Dependencies

```python
required_libraries = {
  "python": [
    "pytesseract",
    "Pillow",
    "opencv-python",
    "fuzzywuzzy",
    "python-Levenshtein"
  ],
  "javascript": [
    "tesseract.js",
    "pdf-lib",
    "canvas"
  ]
}
```

---

12. Security & Compliance

12.1 Data Security

- All invoice files attached to created documents, not OCR entries
- Sensitive data encrypted in database
- User permission-based access control
- Audit trail for all learning modifications

12.2 Error Handling

- Graceful degradation when mandatory fields change
- Automatic retry mechanisms for OCR failures
- Comprehensive logging for debugging

- User-friendly error messages

---

13. Conclusion

This ERPNext OCR Learning App represents a paradigm shift in invoice processing automation. By combining intelligent learning algorithms with comprehensive purchase cycle management, it will achieve unprecedented automation levels while maintaining accuracy and flexibility.

Key Differentiators:

1. **Learning-First Approach:** Every interaction improves future performance
2. **Complete Cycle Management:** From OCR to Payment Entry creation
3. **Dynamic Adaptability:** Handles changing business requirements automatically
4. **Context Intelligence:** Supplier-specific, item-specific, and project-specific learning

Expected ROI:

- 90% reduction in manual data entry
- 95% accuracy in automated processing
- 70% faster invoice processing time
- Elimination of repetitive user interventions

This comprehensive specification provides the foundation for building the most advanced OCR learning system for ERPNext, positioning it as the definitive solution in the market.

Document Version: 1.0
Last Updated: September 2025
Total Pages: 13