Feature #6 Planning Complete! 🎉

Executive Summary

Feature #6: Smart Document Processing & Compliance is now fully planned and ready for implementation!

What This Feature Does

Automates document handling, OCR text extraction, AI-powered field extraction, compliance tracking, and document generation - saving operators 5-10 hours per week on manual data entry.

Business Impact

- 💰 **ROI in 2-3 months** - significant time savings
- ⏱️ **5-10 hours/week saved** per operator
- 📊 **95%+ accuracy** in data extraction
- ✅ **100% compliance tracking** - never miss required documents
- 🚀 **50% faster onboarding** - automated document processing

---

📋 Planning Deliverables

1. Documentation (Complete ✅)

Document	Purpose	Status
`FEATURE_6_DOCUMENT_PROCESSING.md`	Complete feature specification with architecture, timeline, and success metrics	✅ Created
`DOCUMENT_API_SPEC.md`	API endpoint documentation with request/response examples	✅ Created
`FEATURE_6_CHECKLIST.md`	Week-by-week implementation checklist with tasks	✅ Created
`FEATURE_6_READY.md`	Implementation readiness summary	✅ Created

2. Database Design (Complete ✅)

Prisma Schema Updates:
- ✅ `Document` model with OCR and compliance tracking
- ✅ `DocumentTemplate` model for PDF generation
- ✅ `DocumentType`, `ExtractionStatus`, `ComplianceStatus` enums

- ✅ Relations to User, Resident, and Inquiry models
- ✅ Indexes for performance optimization

Migration File:
- ✅ Draft SQL migration created: `prisma/migrations/draft_add_document_processing/migration.sql`

3. Code Structure (Complete ✅)

Type Definitions:
- ✅ `src/types/documents/index.ts` - Complete TypeScript types and interfaces

Utility Libraries:
- ✅ `src/lib/documents/cloudinary.ts` - Cloudinary upload/delete utilities
- ✅ `src/lib/documents/ocr.ts` - OCR text extraction (Tesseract.js, Google Vision)
- ✅ `src/lib/documents/extraction.ts` - AI-powered field extraction (OpenAI GPT-4)
- ✅ `src/lib/documents/classification.ts` - Document type classification
- ✅ `src/lib/documents/compliance.ts` - Compliance checking and tracking
- ✅ `src/lib/documents/generation.ts` - PDF generation from templates

API Structure:

```
src/app/api/documents/
├── upload/          # Document upload endpoint
├── search/          # Document search endpoint
├── compliance/      # Compliance checking endpoint
├── templates/       # Template management endpoints
└── generate/        # Document generation endpoint
```

4. Permissions & RBAC (Complete ✅)

New Permissions Added:
- `DOCUMENTS_VIEW` - View documents
- `DOCUMENTS_CREATE` - Upload documents
- `DOCUMENTS_UPDATE` - Update document metadata
- `DOCUMENTS_DELETE` - Delete documents
- `DOCUMENTS_VIEW_ALL` - View all documents (admin)
- `DOCUMENTS_EXTRACT` - Trigger OCR/AI extraction
- `DOCUMENTS_CLASSIFY` - Classify documents
- `DOCUMENTS_MANAGE_TEMPLATES` - Manage templates (admin)
- `DOCUMENTS_GENERATE` - Generate documents from templates

Role Mappings:
- **ADMIN**: Full access to all document features
- **OPERATOR**: View, create, update, delete, extract, classify, generate
- **CAREGIVER**: View and create documents
- **FAMILY**: View and upload documents for their resident

🏗️ Technical Architecture

System Components

```
┌─────────────────────────────────────────────────────────┐
│                      User Interface                      │
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────┐     │
│  │   Upload     │ │   Viewer     │ │  Compliance  │     │
│  │  Component   │ │  Component   │ │  Dashboard   │     │
│  └──────────────┘ └──────────────┘ └──────────────┘     │
└─────────────────────────────────────────────────────────┘
                          │
                          ▼
┌─────────────────────────────────────────────────────────┐
│                    API Layer (Next.js)                   │
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────┐     │
│  │   Upload     │ │  Extraction  │ │  Generation  │     │
│  │    API       │ │    API       │ │    API       │     │
│  └──────────────┘ └──────────────┘ └──────────────┘     │
└─────────────────────────────────────────────────────────┘
                          │
              ┌───────────┼───────────────┐
              ▼           ▼               ▼
┌──────────────┐ ┌──────────────┐ ┌──────────────┐
│  Cloudinary  │ │   OpenAI     │ │  PostgreSQL  │
│   Storage    │ │   GPT-4      │ │  Database    │
└──────────────┘ └──────────────┘ └──────────────┘
```

Technology Stack

- **Frontend**: React, TypeScript, TailwindCSS
- **Backend**: Next.js API Routes, TypeScript
- **Database**: PostgreSQL with Prisma ORM
- **File Storage**: Cloudinary (already configured)
- **OCR**: Tesseract.js (client-side), Google Cloud Vision API (server-side)
- **AI**: OpenAI GPT-4 (already configured)
- **PDF Generation**: pdf-lib
- **PDF Viewing**: react-pdf

📅 Implementation Timeline

Week 1: Foundation (Dec 19-25, 2025)

Phase 1A: Storage & Database (Days 1-3)
- Run Prisma migration
- Set up Cloudinary integration
- Create upload API endpoint

Phase 1B: Upload UI (Days 4-7)
- Build upload component
- Drag-and-drop functionality
- File validation
- Document list view

Week 2: Extraction (Dec 26-Jan 1, 2026)

Phase 2: OCR (Days 8-10)
- Integrate Tesseract.js
- Google Cloud Vision API setup
- Text extraction API

Phase 3: AI Field Extraction (Days 11-14)
- OpenAI integration for extraction
- Field mapping system
- Form auto-population

Week 3: Classification & Compliance (Jan 2-8, 2026)

Phase 4: Classification (Days 15-17)
- Document type classification
- Search functionality
- Filtering system

Phase 5: Compliance (Days 18-21)
- Compliance checking engine
- Expiration monitoring
- Compliance dashboard

Week 4: Generation & Polish (Jan 9-15, 2026)

Phase 6: Document Generation (Days 22-24)
- Template system
- PDF generation
- Common templates

Phase 7: Testing & Deployment (Days 25-28)
- Comprehensive testing
- Bug fixes
- UI polish
- Documentation
- Deployment

📦 Dependencies to Install

```
# Install required packages
npm install tesseract.js
npm install pdf-lib
npm install react-pdf
npm install @types/pdf-lib
npm install cloudinary

# Optional (for Google Cloud Vision API fallback)
npm install @google-cloud/vision
```

🚀 Getting Started

Step 1: Install Dependencies

```
cd /home/ubuntu/carelinkai-project
npm install tesseract.js pdf-lib react-pdf @types/pdf-lib cloudinary
```

Step 2: Run Database Migration

```
npx prisma migrate dev --name add_document_processing
```

Step 3: Generate Prisma Client

```
npx prisma generate
```

Step 4: Verify Environment Variables

Ensure these are set in Render:
- ✅ `CLOUDINARY_CLOUD_NAME=dygtsnu8z`
- ✅ `CLOUDINARY_API_KEY=328392542172231`
- ✅ `CLOUDINARY_API_SECRET` (existing)
- ✅ `OPENAI_API_KEY` (existing)

Step 5: Start Phase 1A Implementation

Follow the checklist in `docs/features/FEATURE_6_CHECKLIST.md`

✅ Success Criteria

Technical Metrics

- Upload success rate > 99%
- OCR accuracy > 95%
- Field extraction accuracy > 90%
- Classification accuracy > 85%
- Processing time < 30 seconds per document

Business Metrics

- Time saved: 5-10 hours/week per operator
- Data entry errors reduced by 80%
- Compliance rate increased to 100%
- Onboarding time reduced by 50%
- User satisfaction > 4.5/5

📚 Files Created

Documentation Files

```
docs/features/
├── FEATURE_6_DOCUMENT_PROCESSING.md    # Main feature specification
├── DOCUMENT_API_SPEC.md                # API documentation
├── FEATURE_6_CHECKLIST.md              # Implementation checklist
└── FEATURE_6_READY.md                  # Readiness summary
```

Database Files

```
prisma/
├── schema.prisma                                         # Updated with Document
models
└── migrations/draft_add_document_processing/
    └── migration.sql                                     # Draft SQL migration
```

Code Files

```
src/
├── types/documents/
│   └── index.ts                        # TypeScript types and interfaces
├── lib/documents/
│   ├── cloudinary.ts                   # Cloudinary utilities
│   ├── ocr.ts                          # OCR text extraction
│   ├── extraction.ts                   # AI field extraction
│   ├── classification.ts               # Document classification
│   ├── compliance.ts                   # Compliance checking
│   └── generation.ts                   # PDF generation
└── lib/permissions.ts                 # Updated with document permissions
```

Summary Files

```
/home/ubuntu/carelinkai-project/
├── FEATURE_6_PLANNING_COMPLETE.md      # This file
└── prisma/schema_update_documents.txt  # Schema changes reference
```

🎯 Next Actions

1. ✅ **Planning Complete** - All documentation and architecture finalized
2. ⏭️ **Install Dependencies** - Run `npm install` for required packages
3. ⏭️ **Run Migration** - Execute Prisma migration to create tables
4. ⏭️ **Start Phase 1A** - Begin database and storage implementation
5. ⏭️ **Follow Checklist** - Use `FEATURE_6_CHECKLIST.md` for guidance

📞 Support & Resources

- **Feature Overview**: `docs/features/FEATURE_6_DOCUMENT_PROCESSING.md`
- **API Reference**: `docs/features/DOCUMENT_API_SPEC.md`
- **Implementation Guide**: `docs/features/FEATURE_6_CHECKLIST.md`
- **Project Path**: `/home/ubuntu/carelinkai-project`
- **GitHub Repo**: `profyt7/carelinkai`
- **Deployed URL**: `https://carelinkai.onrender.com`

Status: ✅ Planning Complete - Ready to Code!
Created: December 19, 2025
Timeline: 4 weeks (Dec 19, 2025 - Jan 16, 2026)
Expected ROI: 2-3 months

🎉 **All planning is complete! Feature #6 is ready for implementation!**