Wolfstitch Cloud Repository Structure

Foundation for extracting desktop app to cloud-native SaaS

📁 Repository Setup Plan

Repository Strategy

- `wolfstitch-cloud/` (new primary repo for cloud development)
- `wolfcore/` (extracted shared library - will be published to PyPI)

Cloud Repository Structure

```
wolfstitch-cloud/
│
├── 📋 PROJECT FOUNDATION
│   ├── README.md                          # Cloud platform documentation
│   ├── requirements.txt                   # Python dependencies
│   ├── pyproject.toml                     # Modern Python packaging
│   ├── .env.example                       # Environment variables template
│   ├── .gitignore                         # Git ignore patterns
│   ├── docker-compose.yml                # Local development environment
│   └── Dockerfile                         # Production deployment
│
├── 🧠 WOLFCORE LIBRARY (Extracted from Desktop)
│   └── wolfcore/
│       ├── __init__.py                    # Public API exports
│       ├── parsers.py                     # Extracted from: processing/extract.py
│       ├── cleaner.py                     # Extracted from: processing/clean.py
│       ├── chunker.py                     # Extracted from: processing/splitter.py
│       ├── tokenizer_manager.py           # Extracted from: core/tokenizer_manager.py
│       ├── model_database.py              # Extracted from: core/model_database.py
│       ├── processor.py                   # Simplified from: controller.py
│       ├── session_manager.py             # Adapted from: session.py
│       ├── utils.py                       # Shared utilities
│       ├── exceptions.py                  # Custom exception classes
│       └── tests/                         # Comprehensive test suite
│           ├── test_parsers.py
│           ├── test_cleaner.py
│           ├── test_chunker.py
│           ├── test_tokenizer_manager.py
│           └── test_integration.py
│
├── 🚀 BACKEND API (FastAPI)
│   └── backend/
│       ├── main.py                        # FastAPI application entry
│       ├── config.py                      # Configuration management
│       ├── dependencies.py                # Dependency injection
│       └── api/
│           ├── __init__.py
│           ├── auth.py                    # Authentication endpoints
│           ├── files.py                   # File upload/management
│           ├── processing.py              # Job processing endpoints
│           ├── users.py                   # User management
│           └── billing.py                 # Subscription/billing
│       └── models/
```

```
│           ├── __init__.py
│           ├── user.py                    # User data models
│           ├── file.py                    # File data models
│           ├── job.py                     # Processing job models
│           └── schemas.py                 # Pydantic request/response schemas
│       └── services/
│           ├── __init__.py
│           ├── file_service.py            # File handling service
│           ├── processing_service.py      # Job processing orchestration
│           ├── auth_service.py            # Authentication logic
│           └── billing_service.py         # Stripe integration
│       └── workers/
│           ├── __init__.py
│           ├── processing_worker.py       # Background job worker
│           └── tasks.py                   # Celery/RQ tasks
│
├── 🎨 FRONTEND (Next.js)
│   └── frontend/
│       ├── package.json                   # Node.js dependencies
│       ├── next.config.js                 # Next.js configuration
│       ├── tailwind.config.js             # Tailwind CSS setup
│       ├── tsconfig.json                  # TypeScript configuration
│       └── src/
│           ├── app/                       # App Router (Next.js 13+)
│           │   ├── layout.tsx             # Root layout
│           │   ├── page.tsx               # Landing page
│           │   ├── dashboard/             # User dashboard
│           │   ├── auth/                  # Authentication pages
│           │   └── api/                   # API routes (if needed)
│           ├── components/                # Reusable UI components
│           │   ├── ui/                    # shadcn/ui components
│           │   ├── forms/                 # Form components
│           │   ├── layout/                # Layout components
│           │   └── processing/            # Processing-specific components
│           ├── lib/                       # Utility functions
│           │   ├── api.ts                 # API client
│           │   ├── auth.ts                # Auth utilities
│           │   └── utils.ts               # General utilities
│           ├── types/                     # TypeScript type definitions
│           └── styles/                    # Global styles
│
├── 🗄️ DATABASE & INFRASTRUCTURE
│   ├── database/
│   │   ├── migrations/                    # Database migration files
```

```
│   │   ├── seeds/                    # Test data seeds
│   │   └── schema.sql                # Database schema
│   ├── infrastructure/
│   │   ├── terraform/                # Infrastructure as code
│   │   ├── kubernetes/               # K8s deployment configs
│   │   └── monitoring/               # Monitoring configurations
│   └── scripts/
│       ├── setup.sh                  # Development setup script
│       ├── deploy.sh                 # Deployment script
│       └── backup.sh                 # Database backup script
│
└── 📊 DOCUMENTATION & TESTS
    ├── docs/
    │   ├── api/                      # API documentation
    │   ├── deployment/               # Deployment guides
    │   └── user-guide/               # User documentation
    ├── tests/
    │   ├── unit/                     # Unit tests
    │   ├── integration/              # Integration tests
    │   └── e2e/                      # End-to-end tests
    └── monitoring/
        ├── health-checks/            # Health check scripts
        └── performance/              # Performance test scripts
```

🔄 Extraction Mapping

Desktop → Cloud Component Mapping

Desktop File	Cloud Location	Purpose	Extraction Priority
`processing/extract.py`	`wolfcore/parsers.py`	File format parsing	**WEEK 1**
`processing/clean.py`	`wolfcore/cleaner.py`	Text preprocessing	**WEEK 1**
`processing/splitter.py`	`wolfcore/chunker.py`	Text chunking	**WEEK 1**
`controller.py`	`wolfcore/processor.py`	Core orchestration	**WEEK 2**
`session.py`	`wolfcore/session_manager.py`	State management	**WEEK 2**
`core/tokenizer_manager.py`	`wolfcore/tokenizer_manager.py`	Tokenizer handling	**WEEK 3**
`core/model_database.py`	`wolfcore/model_database.py`	AI model specs	**WEEK 3**
`core/cost_calculator.py`	`backend/services/cost_service.py`	Cost analysis	**WEEK 4**
`export/dataset_exporter.py`	`backend/services/export_service.py`	Export functionality	**WEEK 4**

🧠 Wolfcore Library API Design

Simple, Chainable API

```python
# One-line processing (target API)
from wolfcore import Wolfstitch

result = Wolfstitch.process_file(
    "document.pdf",
    tokenizer="gpt-4",
    max_tokens=1024,
    format="jsonl"
)

# Step-by-step processing
wf = Wolfstitch()
parsed = wf.parse("document.pdf")
cleaned = wf.clean(parsed.text, format=parsed.format)
chunks = wf.chunk(cleaned, max_tokens=1024, tokenizer="gpt-4")
result = wf.export(chunks, format="jsonl")

# Async support for cloud
async with Wolfstitch() as wf:
    result = await wf.process_file_async("document.pdf")
```

Key Design Principles

1. **Stateless by default** - No UI dependencies, pure functions
2. **Progressive enhancement** - Basic functionality without premium features
3. **Cloud-ready** - Async support, proper error handling
4. **API compatible** - Easy to wrap in REST endpoints
5. **Testable** - Clear inputs/outputs, mockable dependencies

📋 Week 1 Implementation Checklist

Day 1-2: Repository Foundation

- [ ] Create `wolfstitch-cloud` repository
- [ ] Set up basic directory structure above
- [ ] Initialize Python packaging (`pyproject.toml`, `requirements.txt`)
- [ ] Set up development environment (virtual env, IDE config)
- [ ] Create basic FastAPI skeleton in `backend/main.py`
- [ ] Set up testing framework (pytest configuration)

Day 3-4: First Extraction - File Parsers

- [ ] Extract `processing/extract.py` → `wolfcore/parsers.py`
- [ ] Remove UI dependencies and tkinter imports
- [ ] Create clean, stateless API for file parsing
- [ ] Add comprehensive unit tests
- [ ] Validate PDF, EPUB, TXT parsing still works identically

Day 5-7: Core Processing Pipeline

- [ ] Extract `processing/clean.py` → `wolfcore/cleaner.py`
- [ ] Extract `processing/splitter.py` → `wolfcore/chunker.py`
- [ ] Create unified `wolfcore/processor.py` (simplified controller)
- [ ] Build basic API endpoints in FastAPI
- [ ] End-to-end test: upload → process → download

🎯 Success Criteria for Week 1

Technical:

- [ ] File upload + basic processing working via API
- [ ] Core wolfcore modules extracted and tested
- [ ] Processing output matches desktop app exactly
- [ ] Basic FastAPI deployment working locally

Quality:

- [ ] All files under 600 lines (per your preference)
- [ ] 80%+ test coverage for wolfcore modules
- [ ] Clean, documented APIs
- [ ] No breaking changes to core algorithms

Milestone:

- [ ] Can upload a PDF via API and get back processed chunks
- [ ] Foundation ready for Week 2 tokenizer integration

This structure provides the foundation for rapid cloud development while preserving all the quality and functionality of your desktop app.