PDF Quiz Generator MVP - Detailed Development Plan

Phase 1: Core Pipeline MVP

Project Setup

Environment Requirements

```
Python 3.8+
Pipenv (package and virtual environment manager)
```

Setup with Pipenv

```bash
# Install pipenv if not already installed
pip install pipenv

# Create project directory
mkdir pdf_quiz_mvp
cd pdf_quiz_mvp

# Initialize pipenv environment
pipenv install

# Install required dependencies
pipenv install langchain openai pinecone-client pypdf2 python-dotenv

# Install development dependencies (optional)
pipenv install pytest black --dev

# Activate virtual environment
pipenv shell
```

Pipfile Structure

```toml
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
langchain = "*"
openai = "*"
pinecone-client = "*"
pypdf2 = "*"
python-dotenv = "*"

[dev-packages]
pytest = "*"
black = "*"

[requires]
python_version = "3.8"
```

Environment Variables (.env file)

```
OPENAI_API_KEY=your_openai_key
PINECONE_API_KEY=your_pinecone_key
PINECONE_ENVIRONMENT=your_pinecone_env
PINECONE_INDEX_NAME=pdf-quiz-index
```

Pinecone Setup

Index Configuration

- **Dimension**: 1536 (OpenAI ada-002 embedding dimension)
- **Metric**: Cosine similarity
- **Index Name**: pdf-quiz-index
- **Namespace**: Can use default or create PDF-specific namespaces

Initialization Code Structure

```python
import pinecone
pinecone.init(api_key="your-key", environment="your-env")
index = pinecone.Index("pdf-quiz-index")
```

Core Components Development

1. PDF Text Extraction

Purpose: Load PDF and extract clean text **Implementation**:

- Use LangChain's PyPDFLoader or PDFMinerLoader
- Handle multi-page PDFs
- Clean extracted text (remove excessive whitespace, special characters)
- Validate text extraction success

Error Handling:

- File not found
- Corrupted PDF files
- Empty or image-only PDFs
- Encoding issues

2. Text Chunking Strategy

Purpose: Split large text into manageable, meaningful chunks

Chunking Parameters:

- **Chunk Size**: 200-300 words (optimal for quiz generation)
- **Overlap**: 50 words (maintain context continuity)
- **Separator Strategy**: Prioritize paragraph breaks, then sentences

Implementation Approach:

- Use LangChain's RecursiveCharacterTextSplitter
- Preserve sentence boundaries
- Include metadata (chunk_id, source_page, position)

3. Embedding Generation

Purpose: Convert text chunks to vector representations

Configuration:

- **Model**: OpenAI text-embedding-ada-002
- **Batch Processing**: Process multiple chunks efficiently
- **Rate Limiting**: Handle API rate limits gracefully

Metadata Storage:

- Chunk text content
- Source PDF information
- Page numbers
- Chunk position/sequence

4. Pinecone Vector Storage

Purpose: Store embeddings with searchable metadata

Storage Strategy:

- Unique IDs for each chunk
- Comprehensive metadata for filtering
- Namespace organization (optional for MVP)

Data Structure:

python

```python
{
    "id": "pdf_name_chunk_001",
    "values": [embedding_vector],
    "metadata": {
        "text": "chunk_content",
        "source": "document.pdf",
        "page": 1,
        "chunk_index": 0
    }
}
```

5. Semantic Search & Retrieval

Purpose: Find most relevant chunks for quiz generation

Query Strategies:

- **Primary Query**: "important concepts definitions key facts"
- **Secondary Query**: "main principles theories critical information"
- **Fallback Query**: "significant details noteworthy points"

Retrieval Parameters:

- **Top-K**: 3-4 chunks for small PDFs
- **Similarity Threshold**: 0.7+ (adjust based on testing)

- **Diversity**: Ensure chunks from different sections

6. Quiz Generation via OpenAI

Purpose: Generate high-quality multiple-choice questions

Prompt Engineering:

```
You are an expert quiz creator. Generate multiple-choice questions from the
following content.

Requirements:
- Create 5-8 questions
- Each question should have 4 options (A, B, C, D)
- Only one correct answer per question
- Focus on key concepts and important facts
- Vary difficulty levels
- Include the correct answer

Content: {retrieved_chunks}

Format your response as:
Question 1: [question text]
A) [option]
B) [option]
C) [option]
D) [option]
Correct Answer: [letter]

[Continue for all questions]
```

Generation Parameters:

- **Model**: GPT-3.5-turbo or GPT-4
- **Temperature**: 0.3 (for consistency)
- **Max Tokens**: 1500-2000

MVP Script Structure

Main Script Flow

```python
def main():
    # 1. Initialize services
    setup_openai()
    setup_pinecone()

    # 2. Process PDF
    pdf_path = "path/to/your/document.pdf"
    text = extract_pdf_text(pdf_path)

    # 3. Create and store chunks
    chunks = create_chunks(text)
    embeddings = generate_embeddings(chunks)
    store_in_pinecone(embeddings, chunks)

    # 4. Retrieve relevant content
    relevant_chunks = query_pinecone_for_quiz_content()

    # 5. Generate quiz
    quiz = generate_quiz_from_chunks(relevant_chunks)

    # 6. Display results
    display_quiz(quiz)
```

Function Specifications

extract_pdf_text(pdf_path)

- Input: File path string
- Output: Cleaned text string
- Error handling for file access and PDF parsing

create_chunks(text)

- Input: Full text string
- Output: List of text chunks with metadata
- Implement overlap and size controls

generate_embeddings(chunks)

- Input: List of text chunks
- Output: List of embedding vectors
- Batch processing for efficiency

store_in_pinecone(embeddings, chunks)

- Input: Embeddings and corresponding chunks
- Output: Confirmation of successful storage
- Handle duplicate detection

query_pinecone_for_quiz_content()

- Input: None (uses predefined queries)
- Output: Most relevant chunks for quiz generation
- Multiple query strategy implementation

generate_quiz_from_chunks(chunks)

- Input: Retrieved text chunks
- Output: Formatted quiz questions
- Structured prompt and response parsing

Testing & Validation

Test Cases

1. **Small PDF (500-1000 words)**
    - Verify all chunks are processed
    - Check embedding quality
    - Validate quiz question relevance
2. **Medium PDF (2000-3000 words)**
    - Test chunking strategy effectiveness
    - Ensure diverse content retrieval
    - Verify quiz covers different sections
3. **Technical PDF**
    - Test with domain-specific content
    - Validate terminology handling
    - Check question complexity appropriateness

Success Metrics

- **Processing Speed**: Complete workflow under 2 minutes
- **Quiz Quality**: Questions clearly answerable from content
- **Content Coverage**: Questions from different PDF sections

- **Accuracy**: No hallucinated information in questions

Error Handling & Edge Cases

Common Issues

- **Empty PDF or extraction failure**
- **Very short documents (< 200 words)**
- **Highly technical content with specialized terms**
- **API rate limits or timeouts**
- **Pinecone connection issues**

Graceful Degradation

- Fallback to full-text processing if vector search fails
- Reduce chunk requirements if insufficient content
- Provide informative error messages
- Log errors for debugging

Output Format

Running the MVP

```bash
# Method 1: Activate shell and run
pipenv shell
python main.py

# Method 2: Run directly without shell activation
pipenv run python main.py

# Method 3: Run with specific Python version
pipenv run python3 main.py
```

```
=== PDF Quiz Generator ===
Processing: document.pdf
✓ Text extracted (1,247 words)
✓ Created 5 chunks
✓ Generated embeddings
✓ Stored in Pinecone
✓ Retrieved 3 relevant chunks
✓ Generated quiz

=== QUIZ QUESTIONS ===

Question 1: What is the primary function of...
A) Option 1
B) Option 2
C) Option 3
D) Option 4
Correct Answer: B

[Continue for all questions]

=== QUIZ COMPLETE ===
Generated 6 questions from PDF content
```

File Structure

```
pdf_quiz_mvp/
├── Pipfile              # Dependencies specification
├── Pipfile.lock         # Locked dependency versions
├── main.py              # Main script
├── config.py           # Configuration and constants
├── pdf_processor.py    # PDF handling functions
├── vector_manager.py   # Pinecone operations
├── quiz_generator.py   # OpenAI quiz generation
├── .env               # Environment variables
└── sample_pdfs/       # Test documents
```

Console Display

Next Steps After MVP Success

1. Add configuration options (chunk size, question count)
2. Implement multiple PDF processing
3. Add quiz export functionality (JSON, text file)
4. Create simple CLI interface with arguments

5. Prepare for API wrapper development