CourseMaster AI - Engineering Report

NLP Course Project - Development Track
Date: December 2024
Version: 2.0.0

---

1. Problem Statement

Target Users

Students preparing for exams, self-learners, and professionals seeking personalized study assistance from their own documents.

Pain Points

- **Lack of Personalization**: Generic study materials don't adapt to individual weaknesses
- **Manual Quiz Creation**: Time-consuming to create practice questions from PDFs
- **No Progress Tracking**: Difficult to identify which concepts need more focus
- **Limited Interaction**: Static PDFs offer no way to ask clarifying questions

Solution

CourseMaster AI transforms static PDF documents into an interactive learning experience using AI. The platform: 1. Extracts knowledge from user-uploaded PDFs using RAG 2. Generates personalized quizzes at multiple difficulty levels 3. Provides real-time Q&A with source citations 4. Tracks weak concepts for targeted improvement

Market Fit

- **EdTech Growth**: $340B market (2024)
- **AI Tutoring**: 40% CAGR through 2030
- **Differentiation**: Multi-user RAG platform with LangChain agents

---

2. System Architecture

Technology Stack

Agentic AI Framework (Required) - **LangChain 0.1.0**: Modern agentic pipeline - `RetrievalQA` chain for RAG-based Q&A - `ConversationalRetrievalChain` for chat with memory - Agent system for structured quiz generation

Backend - FastAPI 0.109+ (async Python web framework) - SQLAlchemy 2.0 (ORM) - PostgreSQL via Supabase (cloud database) - JWT authentication + bcrypt password hashing

AI/ML - Groq API (LLaMA 3.3 70B model) - ChromaDB 0.4.22 (vector database) - Sentence Transformers (all-MiniLM-L6-v2 embeddings) - PyPDF for text extraction

Frontend - React 18 + Vite - Tailwind CSS - React Router v6 - Axios for API calls

DevOps - Docker + Docker Compose - Pytest (automated testing) - Structured JSON logging - Git version control

Component Diagram

See ARCHITECTURE.md for detailed Mermaid diagram.

Key Components: 1. **Frontend** → User interface 2. **FastAPI Server** → REST API 3. **LangChain Service** → Agent orchestration 4. **RAG Service** → Document processing & retrieval 5. **ChromaDB** → Vector embeddings 6. **PostgreSQL** → User data & metadata 7. **Groq API** → LLM inference

---

3. Technical Implementation

3.1 RAG Pipeline

Document Processing:

```
1. User uploads PDF → PyPDF extracts text
2. Text chunked (512 tokens, 50 overlap)
3. Sentence Transformers generate embeddings
4. ChromaDB stores vectors with metadata
5. PostgreSQL stores PDF metadata
```

Retrieval: - Cosine similarity search in ChromaDB - Top-k relevant chunks (k=5 for chat, k=15 for quiz) - LangChain retriever integration

3.2 LangChain Agentic Framework

RetrievalQA Chain (Chat):

```
langchain_service.create_rag_chain(collection_name)
→ Chroma vectorstore.as_retriever()
→ Custom prompt template
→ RetrievalQA.from_chain_type()
→ Returns answer + source documents
```

Agent-Based Quiz Generation:

```
langchain_service.generate_quiz_with_agent()
→ ChatGroq LLM
→ Structured prompt with difficulty guidelines
→ JSON output parsing
→ Returns list of questions with concepts
```

3.3 Multi-User Architecture

Data Isolation: - Every query filtered by `user_id` - ChromaDB collections: `user_{id}_file_{file_id}` - JWT tokens for authentication - Row-level security in PostgreSQL

Authentication Flow:

```
1. Register → Hash password (bcrypt) → Store user
2. Login → Verify password → Generate JWT
3. Protected routes → Verify JWT → Get user_id
```

3.4 API Design

RESTful Endpoints: - `POST /api/auth/register` - Create account - `POST /api/auth/login` - Get JWT token - `POST /api/pdf/upload` - Upload & index PDF - `POST /api/chat/ask` - RAG-based Q&A - `POST /api/quiz/generate` - Generate quiz (easy/medium/hard) - `POST /api/quiz/submit` - Submit & grade quiz - `GET /api/analytics/weaknesses` - Get tracked concepts

Request/Response Format: - Input: JSON (Pydantic validated) - Output: JSON with proper HTTP status codes - Errors: Structured error responses

3.5 Testing Strategy

Unit Tests (`tests/test_auth.py`, `tests/test_pdf.py`): - Registration/login flows - PDF upload validation - JWT token verification

Integration Tests (`tests/test_integration.py`): - End-to-end user workflows - Health check endpoints

Test Coverage: - Pytest with coverage reports - Test fixtures for database & auth - Run: `pytest --cov`

Configuration: `pytest.ini` with async support

3.6 Logging & Monitoring

Structured Logging:

```
logging_config.setup_logging(level="INFO")
→ JSON format logs
→ Console + File handlers
```

→ Request/response middleware
→ Error tracking with stack traces

Log Files: - `logs/app.log` - All logs (rotated at 10MB) - `logs/error.log` - Errors only

Log Format:

```
{
  "timestamp": "2024-12-28T14:30:00Z",
  "level": "INFO",
  "message": "Quiz generated",
  "user_id": "123",
  "duration_ms": 1250
}
```

---

4. Deployment

Local Development

Backend:

```
cd backend
pip install -r requirements.txt
python -m uvicorn main:app --reload
```

Frontend:

```
cd frontend
npm install
npm run dev
```

Docker Deployment

Single Command:

```
docker-compose up -d
```

Services: 1. ChromaDB (port 8001) 2. Backend (port 8000) 3. Frontend (port 3000)

Environment Variables: - `DATABASE_URL` - PostgreSQL connection - `GROQ_API_KEY` - Groq API key - `SECRET_KEY` - JWT secret

Production Considerations

- **Database**: Managed PostgreSQL (Supabase)
- **Logging**: Centralized log aggregation
- **Secrets**: Environment-based configuration
- **Scaling**: Horizontal scaling with load balancer

- **Monitoring**: Health check endpoints

---

5. Key Features

Multi-User System - Complete data isolation
RAG Pipeline - ChromaDB + Sentence Transformers
LangChain Agents - Modern agentic framework
Adaptive Quizzes - 3 difficulty levels
Weakness Tracking - AI-powered concept extraction
Chat with Memory - Conversational Q&A
Production-Ready - Docker, logging, testing

---

6. Testing & Quality

- **Automated Tests**: Pytest suite with 10+ tests
- **Test Coverage**: Unit + integration tests
- **API Validation**: Pydantic models
- **Logging**: Structured JSON logs
- **Error Handling**: Comprehensive exception handling
- **Documentation**: OpenAPI/Swagger docs at `/docs`

---

7. Future Enhancements

Priority 2 Features (if time permits): - Multi-agent system (retriever + grader + analyzer) - Prometheus + Grafana monitoring - CI/CD pipeline (GitHub Actions) - Advanced RAG (hybrid search, re-ranking) - Flashcard generation system

---

8. Repository

GitHub: https://github.com/I221165/NLP-Project

Structure:

```
NLP-Project/
  backend/          # FastAPI server
     routers/     # API endpoints
     services/    # LangChain, RAG, Groq
     database/    # Models, connection
     tests/       # Pytest suite
     logs/        # Application logs
```

```
frontend/          # React application
docs/             # Documentation
docker-compose.yml
README.md
```

---

9. Conclusion

CourseMaster AI demonstrates a production-grade AI system using modern frameworks (LangChain), advanced techniques (RAG), and engineering best practices (testing, logging, containerization). The platform successfully addresses real-world learning challenges through personalized, AI-powered tutoring from user-provided documents.

Course Requirements Met: - Agentic pipeline (LangChain) - RAG with vector store (ChromaDB) - Model integration (Groq/LLaMA) - REST API (FastAPI) - Dockerization - Automated testing - Logging & monitoring - Git workflow

---

Total Pages: 3
Report Type: Engineering-focused
Submission Ready: