AI Lecture Highlight Extractor

Project Overview

A multi-agent AI pipeline that transforms long lecture recordings into concise, engaging highlight videos. The system processes uploaded lecture videos through intelligent agents to identify key moments, remove silent/blank sections, and create a polished 2-minute summary video.

Value Proposition

- **Input**: Long lecture recording (30+ minutes) with shared screen + lecturer camera
- **Output**: Multiple short highlight videos (20 seconds to 2 minutes each) featuring key concepts
- **Video Processing**: Intelligent screen splitting (shared content | lecturer view) with optimized layouts
- **Subtitles**: Auto-generated captions overlaid on each video segment
- **Time Savings**: Automatically identifies and segments important moments, removes dead time
- **Quality Enhancement**: Professional video formatting with synchronized audio and visual elements
- **Educational Focus**: Optimized for academic content with topic-based segmentation

Tech Stack

Backend Framework

- **Python 3.9+** - Main programming language
- **FastAPI** - Modern, fast web framework with automatic API documentation
- **Celery** - Distributed task queue for agent coordination
- **Redis** - Message broker and caching
- **Pydantic** - Data validation and serialization

Frontend Framework

- **React 18+** - Modern UI library with hooks
- **TypeScript** - Type-safe JavaScript
- **Vite** - Fast build tool and dev server
- **Tailwind CSS** - Utility-first CSS framework
- **React Query** - Data fetching and state management

- **Socket.io Client** - Real-time updates

AI/ML Libraries

- **Google Gemini API** - Content analysis and importance scoring
- **Google AI Python SDK** - Official Python client for Gemini API integration
- **OpenAI Whisper** - Speech-to-text transcription + subtitle generation
- **MoviePy** - Video processing, screen splitting, and multi-segment compilation
- **OpenCV** - Computer vision for layout detection and screen region identification
- **PyDub** - Audio processing and silence detection
- **scikit-learn** - Machine learning for segment classification
- **NumPy** - Numerical computations for audio/video analysis
- **librosa** - Audio analysis and feature extraction
- **Pillow (PIL)** - Image processing for subtitle overlay and layout optimization

Cloud Storage & Infrastructure

- **AWS S3** - Video and media file storage
- **AWS CloudFront** - CDN for fast media delivery
- **PostgreSQL** - Production database (local SQLite for development)
- **Docker** - Containerization
- **Docker Compose** - Local development orchestration

Deployment Options

- **AWS ECS/Fargate** - Container orchestration
- **Vercel** - Frontend deployment (React app)
- **Railway/Render** - Backend API deployment alternative
- **AWS Lambda** - Serverless functions for lightweight processing

Video Storage Strategy

Development Environment

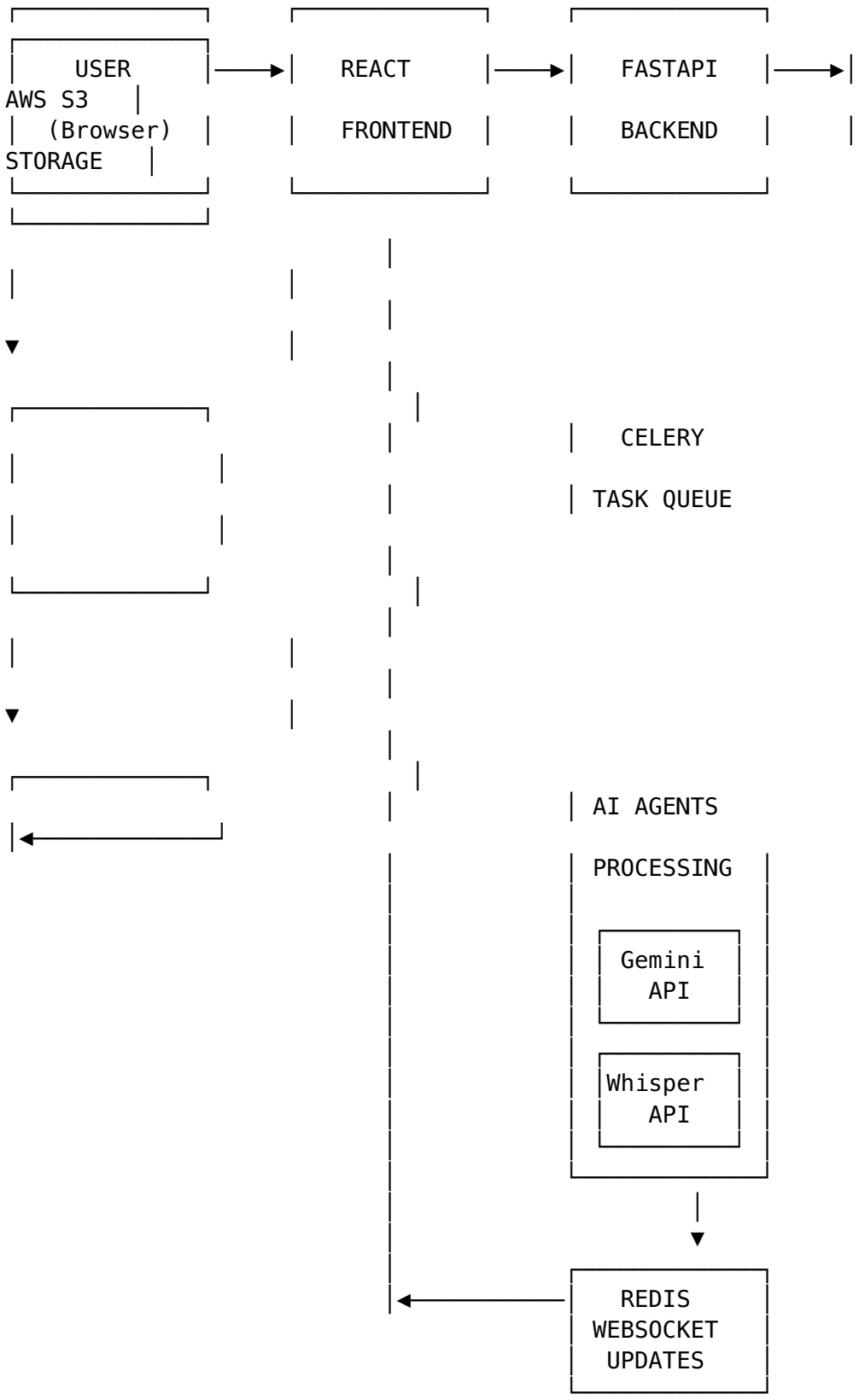
- **Local File System** - Quick testing and development
- **Temporary Storage** - Videos deleted after processing completion

Production Environment

- **AWS S3** - Primary video storage with lifecycle policies
- **Retention Policy**:
  - Videos stored for 7 days after processing
  - Automatic deletion via S3 lifecycle rules
  - Generated content (thumbnails, clips) kept for 30 days
- **CDN**: CloudFront for fast global access to generated media
- **Security**: Pre-signed URLs for secure upload/download

Storage Architecture

System Architecture Flow



Upload Flow: User uploads via React App → FastAPI generates S3 pre-signed URL → Direct S3 upload → Celery Task Queue processes video → Gemini API + Whisper process content → Generated content stored back to S3

Project Structure

Backend Structure

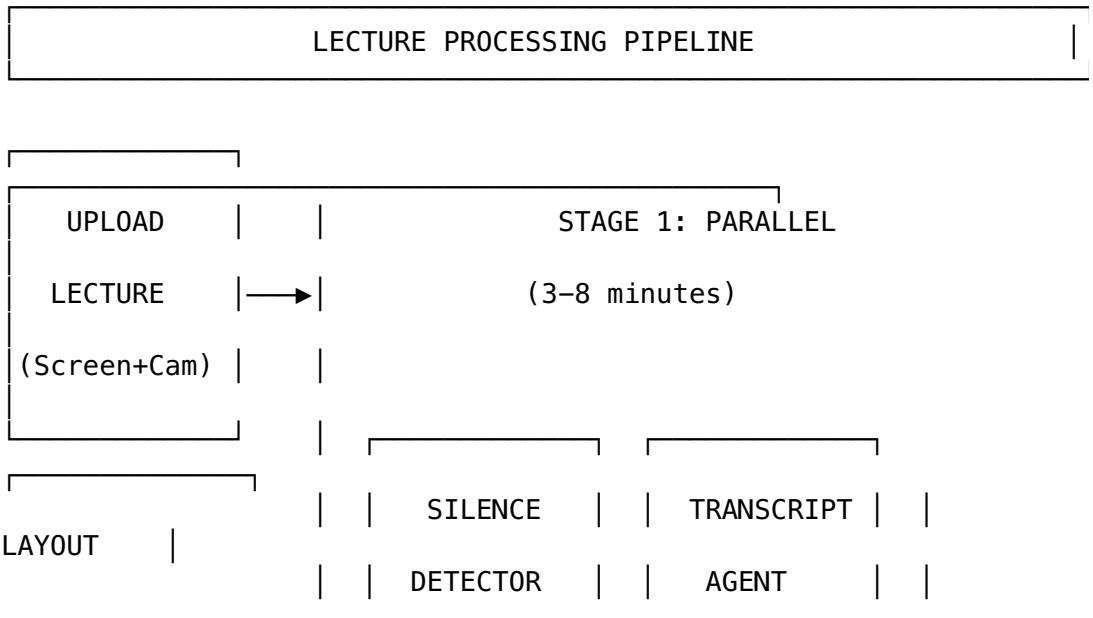
- **app/**: Main FastAPI application
  - **main.py**: FastAPI application entry point
  - **api/routes/**: API endpoint definitions (upload, jobs, results)
  - **models/**: Pydantic data models
  - **services/**: Business logic (S3 operations, database)
  - **core/**: Configuration and security
- **agents/**: AI processing agents (clipper, caption, thumbnail, editor, copywriter, scheduler)
- **pipeline/**: Workflow orchestration and Celery tasks

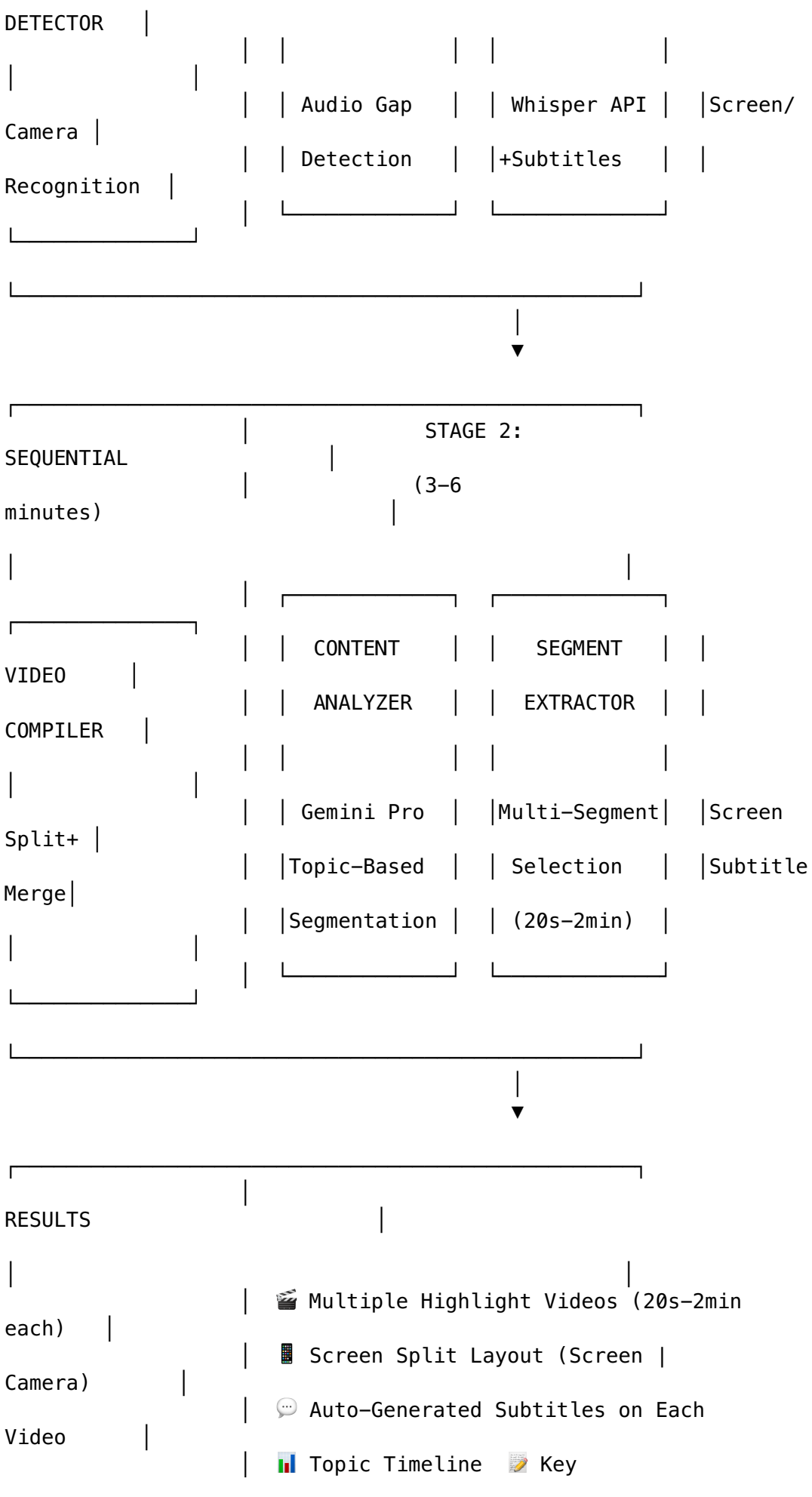
Frontend Structure

- **src/components/**: React components organized by feature
  - **Upload/**: Video upload interface
  - **Dashboard/**: Real-time processing status
  - **Results/**: Campaign results display
- **src/hooks/**: Custom React hooks for API calls and WebSocket
- **src/services/**: API client and WebSocket management
- **src/types/**: TypeScript type definitions

Agent Architecture

Visual Workflow Overview




DETECTOR
Audio Gap Detection
Whisper API +Subtitles
Screen/ Camera Recognition
STAGE 2: SEQUENTIAL (3-6 minutes)
CONTENT ANALYZER
Gemini Pro Topic-Based Segmentation
SEGMENT EXTRACTOR
Multi-Segment Selection (20s-2min)
VIDEO COMPILER
Screen Split+ Subtitle Merge
RESULTS
Multiple Highlight Videos (20s-2min each)
Screen Split Layout (Screen | Camera)
Auto-Generated Subtitles on Each Video
Topic Timeline
Key

```
Concepts              |
└──────────────────────────────────────────────────┘
```

Parallel Processing Stage (Stage 1)

Three agents run simultaneously on the uploaded lecture:

1. **Silence Detector Agent** - Identifies silent gaps, long pauses, and blank sections in audio
2. **Transcript Agent** - Transcribes speech to text with precise timestamps + generates subtitle files using OpenAI Whisper
3. **Layout Detector Agent** - Uses OpenCV to detect screen sharing regions vs lecturer camera, identifies optimal split layouts

Sequential Processing Stage (Stage 2)

Three agents run in sequence using parallel stage outputs:

1. **Content Analyzer Agent** - Uses Gemini Pro to analyze transcript and segment by topics/concepts (20s-2min chunks)
2. **Segment Extractor Agent** - Creates multiple highlight segments, scores by importance, removes silence/blanks
3. **Video Compiler Agent** - For each segment: applies screen split layout, overlays subtitles, renders final videos with consistent formatting

API Design

Core Endpoints

- **Upload Management**: Pre-signed URL generation for lecture uploads, processing initiation
- **Job Management**: Real-time status, progress tracking, processing logs for highlight extraction
- **Results**: Highlight video retrieval, segment analysis, timeline summary
- **WebSocket**: Real-time updates during lecture processing and video compilation

User Interface Design

Upload Page

- **Drag-and-drop Lecture Upload**: Intuitive file selection with video format validation
- **Lecture Configuration**: Lecture title, subject/topic selection, duration settings
- **Progress Indication**: Upload progress with real-time feedback

Processing Dashboard

- **Real-time Progress**: Live updates on silence detection, transcription, and highlight extraction
- **Agent Visualization**: Clear indication of parallel analysis vs sequential compilation
- **Estimated Completion**: Dynamic time remaining for highlight generation
- **Error Handling**: User-friendly error messages and retry options

Results Page

- **Multiple Video Gallery**: Collection of highlight videos (20s-2min each) with topic labels
- **Screen Split Preview**: Each video shows optimized screen/camera layout
- **Subtitle Preview**: Toggle subtitles on/off for each video segment
- **Segment Analysis**: Timeline showing topic segmentation and importance scores
- **Export Options**: Download individual videos or complete package in multiple formats

Main Workflow After Lecture Upload

1. Frontend Upload Process

User selects lecture video file, React app requests pre-signed S3 URL from FastAPI backend, direct upload to S3 with progress tracking, backend triggered for lecture processing pipeline initiation.

2. Backend Processing Initiation

FastAPI receives lecture processing request, creates job record in database, triggers Celery pipeline with S3 lecture location, returns job ID for tracking.

3. Parallel Agent Execution (3-8 minutes)

Celery orchestrator downloads lecture from S3 and triggers parallel processing:

Silence Detector Agent: Analyzes audio waveform for silent gaps and pauses, uses PyDub and librosa for amplitude analysis, identifies sections with minimal audio activity, creates timestamp map of silent vs active regions.

Transcript Agent: Extracts audio track from lecture video, sends audio to OpenAI Whisper API, processes transcription with precise timestamps, generates subtitle files (SRT/VTT) with proper formatting, stores transcript with speaker segment metadata.

Layout Detector Agent: Uses OpenCV to analyze video frames and detect screen sharing vs camera regions, identifies consistent layout patterns throughout lecture, determines optimal split ratios for screen content and lecturer view.

4. Sequential Agent Execution (2-4 minutes)

After parallel completion, sequential processing begins:

Content Analyzer Agent: Uses Gemini Pro to analyze transcript and segment by educational topics/concepts, creates 20-second to 2-minute topic-based chunks, scores each segment by importance and educational value, identifies key learning moments and transitions.

Segment Extractor Agent: Combines silence detection, layout analysis, and content scoring to create multiple highlight segments, removes blank/silent sections from each segment, ensures each video maintains narrative coherence and educational flow.

Video Compiler Agent: For each segment: uses MoviePy to extract video with screen split layout, overlays generated subtitles with proper styling, applies consistent branding and formatting, renders multiple final videos with optimized encoding.

5. Result Presentation

Dashboard updates with completion status, results page becomes available with multiple video gallery showing all highlight segments, each video displays with screen split layout and subtitle controls, segment analysis shows topic breakdown and importance timeline, user can preview individual videos or download complete package.

Key Features

Real-Time Processing Updates

- WebSocket connection provides live progress updates during lecture analysis
- Visual indicators show current agent status and highlight extraction progress
- Error handling with detailed logging and user-friendly messages

Intelligent Content Extraction

- AI-powered silence detection eliminates dead time and blank sections
- Advanced transcript analysis identifies key educational moments
- Smart segment scoring ensures most important content is preserved

User Experience Optimization

- Intuitive drag-and-drop lecture upload interface
- Clear progress indication throughout highlight extraction
- Interactive results with timeline visualization and key topics
- Mobile-responsive design optimized for educational content viewing

Performance Considerations

Processing Optimization

- Parallel audio/visual analysis reduces processing time by 60-70%
- S3 direct upload eliminates server bandwidth bottlenecks for large lecture files
- Celery distributed processing enables horizontal scaling for multiple lectures
- Smart segment caching prevents reprocessing of similar educational content

Infrastructure Scaling

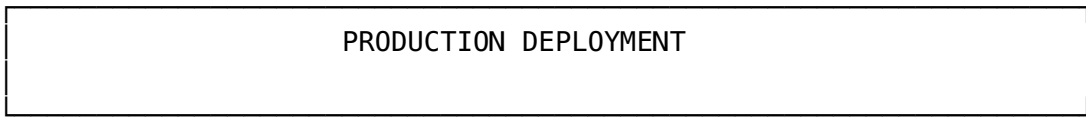
- Docker containerization for consistent deployment
- Separate frontend and backend scaling
- Redis clustering for high-availability message queuing
- CloudFront CDN for global content delivery

Deployment Architecture

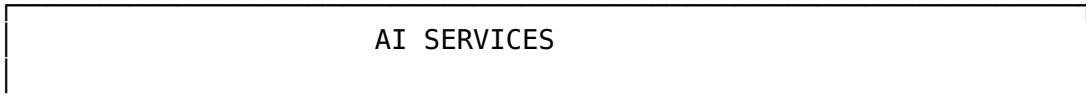
Development Environment

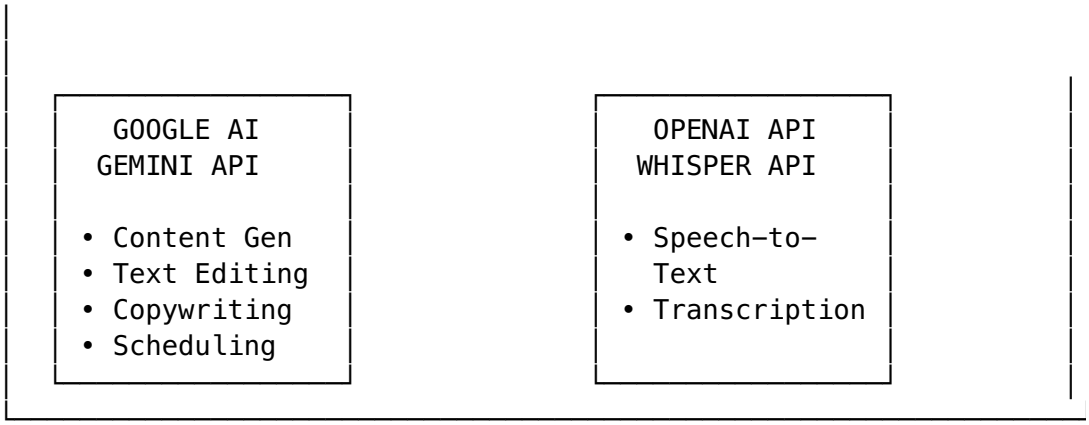
```
docker-compose up:
├── FastAPI Backend (localhost:8000)
├── React Frontend (localhost:3000)
├── Redis (localhost:6379)
├── Celery Worker
└── PostgreSQL (localhost:5432)
```

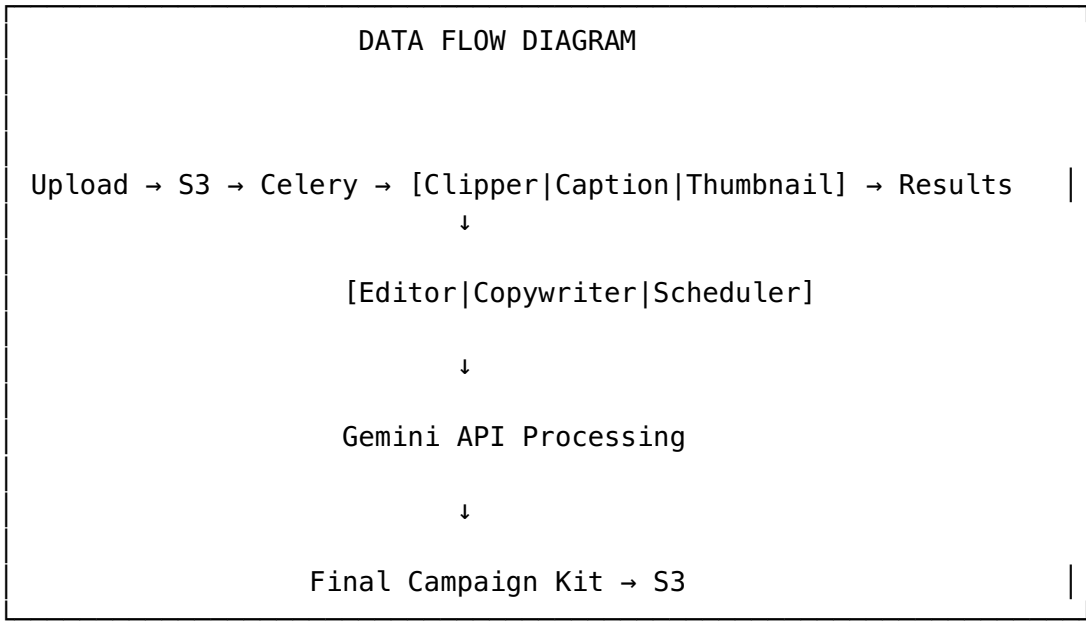
Production Environment




```
Frontend: Vercel/Netlify (React build)
     ↓
Backend API: AWS ECS/Railway (FastAPI containers)
     ↓
Database: AWS RDS PostgreSQL
     ↓
Queue: AWS ElastiCache Redis
     ↓
Storage: S3 + CloudFront CDN
     ↓
Workers: ECS Fargate (Celery containers)
     ↓
```







Security Considerations

- Pre-signed URLs with expiration for secure S3 access
- Input validation and file type restrictions
- Rate limiting on API endpoints
- CORS configuration for frontend-backend communication
- Environment variable management for sensitive credentials

Demo Scenarios

Test Lecture Recommendations

- **Computer Science Lecture** (45 minutes): Demonstrates technical content processing
- **History Class** (30 minutes): Shows narrative content highlight extraction
- **Math Tutorial** (60 minutes): Tests problem-solving segment identification

Expected Performance

- **Processing Time**: 8-15 minutes for 30-60 minute input lecture
- **Output Quality**: Multiple polished highlight videos (3-8 segments) with screen split + subtitles
- **Content Preservation**: 90%+ of important educational moments captured across all segments
- **Video Output**: 3-8 topic-based videos ranging from 20 seconds to 2 minutes each

This system demonstrates modern AI pipeline architecture while providing practical value for educators, students, and educational content creators looking to make lectures more accessible and engaging.