

EduFlow AI - Implementation Plan

Cloud-Based Educational AI Platform with SOLID Principles

Table of Contents

1. [Executive Summary](#)
2. [Architecture Overview](#)
3. [SOLID Principles Application](#)
4. [Phase-by-Phase Implementation](#)
5. [Dependencies & Setup](#)
6. [API Integration Strategy](#)
7. [Frontend Architecture](#)
8. [Testing Strategy](#)

Executive Summary

What We're Building

A cloud-based AI study companion that transforms course materials (PDFs, slides, videos) into structured study resources (notes, flashcards, quizzes, slides) using specialized AI agents.

Key Architectural Features

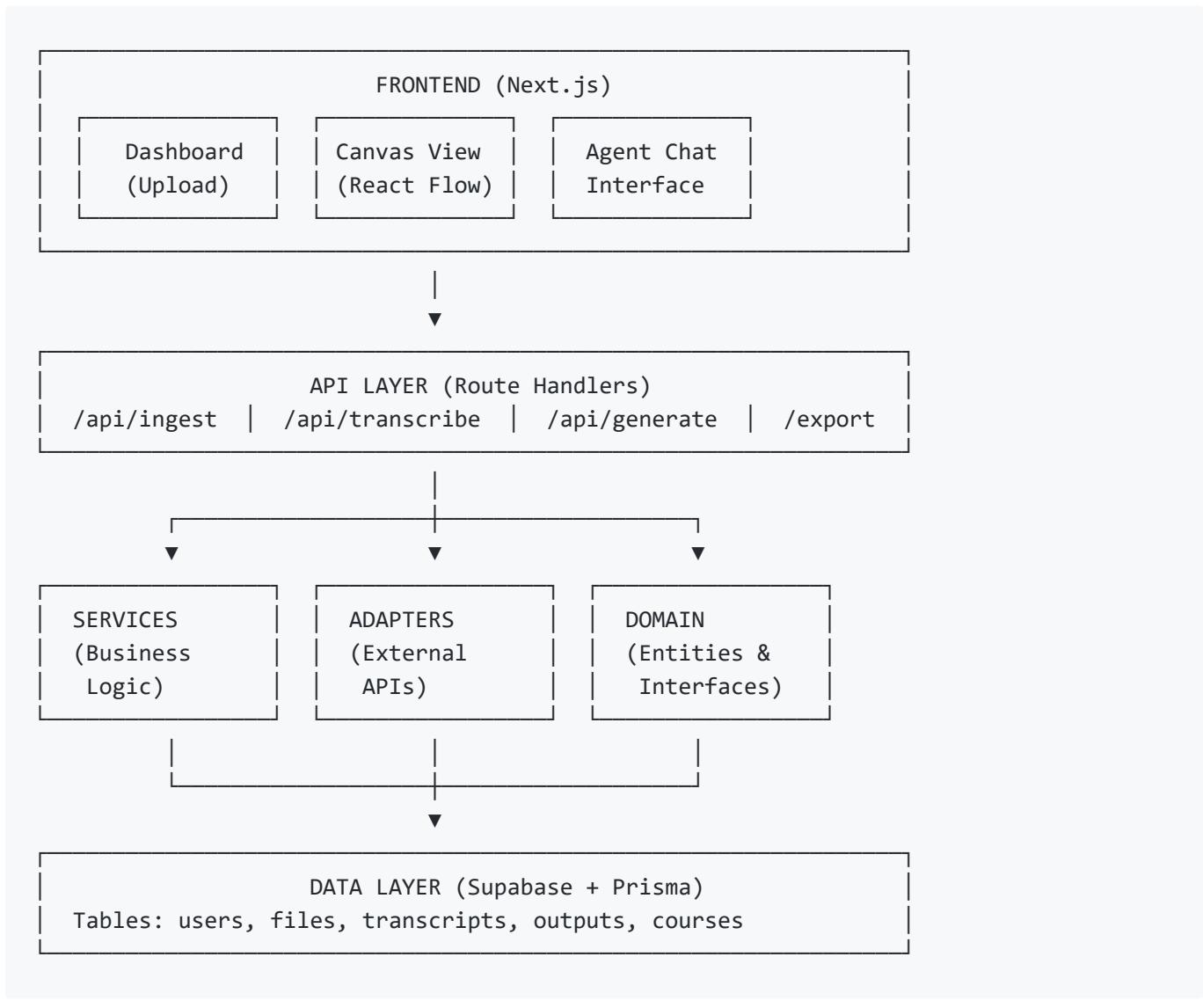
- Infinite Canvas UX: Spatial organization of AI agents and outputs
- Agent-Based Architecture: Specialized AI workers for different tasks
- Real-time Processing: Async job queue with progress indicators
- Export Pipeline: Multi-format output generation

Our Approach

- Educational Focus: Course material ingestion + Canvas LMS sync
- SOLID Architecture: Clean separation of concerns, DI, ISP
- Multiple AI Providers: Gemini (primary), OpenRouter (fallback)

- Self-Hosted Whisper: Cost-effective transcription on Digital Ocean
-

II Architecture Overview



⌚ SOLID Principles Application

1 Single Responsibility Principle (SRP)

Each class/module has ONE reason to change

```

// ✗ BAD: God class doing everything
class AIService {
  uploadFile() { }
  transcribeVideo() { }
  generateNotes() { }
  exportPDF() { }
}

// ✓ GOOD: Separate services
class IngestService { processUpload() { } }
class TranscribeService { transcribeAudio() { } }
class GenerateService { generateContent() { } }
class ExportService { exportToPDF() { } }

```

2 Open/Closed Principle (OCP)

Open for extension, closed for modification

```

// Base agent interface
interface IAgent {
  name: string;
  process(input: ProcessInput): Promise<ProcessOutput>;
}

// Extend without modifying existing code
class NotesAgent implements IAgent { }
class FlashcardAgent implements IAgent { }
class QuizAgent implements IAgent { }
class SlidesAgent implements IAgent { }

```

3 Liskov Substitution Principle (LSP)

Subtypes must be substitutable for their base types

```

// Any IModelClient can be swapped
interface IModelClient {
  complete(prompt: string, options?: CompletionOptions): Promise<string>;
}

class GeminiClient implements IModelClient { }
class OpenRouterClient implements IModelClient { }
class ClaudeClient implements IModelClient { }

// Usage - can swap clients without breaking code
function processWithAI(client: IModelClient, prompt: string) {
  return client.complete(prompt);
}

```

4 Interface Segregation Principle (ISP)

Clients shouldn't depend on interfaces they don't use

```

// ✗ BAD: Fat interface
interface IAgent {
  generateNotes(): Promise<string>;
  generateFlashcards(): Promise<Flashcard[]>;
  generateQuiz(): Promise<Quiz>;
  generateSlides(): Promise<Slide[]>;
}

// ✅ GOOD: Segregated interfaces
interface INotesGenerator {
  generateNotes(): Promise<string>;
}

interface IFlashcardGenerator {
  generateFlashcards(): Promise<Flashcard[]>;
}

interface IQuizGenerator {
  generateQuiz(): Promise<Quiz>;
}

```

5 Dependency Inversion Principle (DIP)

Depend on abstractions, not concretions

```
// ❌ BAD: Direct dependency
class GenerateService {
  private gemini = new GeminiClient(); // Tight coupling
}

// ✅ GOOD: Dependency injection
class GenerateService {
  constructor(private modelClient: IModelClient) { }
}

// Inject at runtime
const geminiClient = new GeminiClient(process.env.GEMINI_API_KEY);
const service = new GenerateService(geminiClient);
```

Phase-by-Phase Implementation

Phase 1: Foundation & Setup (Days 1-2)

Goal: Set up project infrastructure and core dependencies

Tasks:

1. Environment Setup

- Create `.env.local` with all API keys
- Install core dependencies
- Configure TypeScript, ESLint, Prettier

2. Database Schema (Prisma)

```

model User {
    id      String  @id @default(cuid())
    email   String  @unique
    auth0Id String  @unique
    courses Course[]
    createdAt DateTime @default(now())
}

model Course {
    id      String  @id @default(cuid())
    name    String
    userId  String
    user    User     @relation(fields: [userId], references: [id])
    files   File[]
}

model File {
    id      String      @id @default(cuid())
    name    String
    type    String      // pdf, pptx, mp4, docx
    url    String
    courseId String
    course  Course     @relation(fields: [courseId], references: [id])
    transcripts Transcript[]
    createdAt DateTime    @default(now())
}

model Transcript {
    id      String  @id @default(cuid())
    content String  @db.Text
    fileId  String
    file    File     @relation(fields: [fileId], references: [id])
    outputs Output[]
    createdAt DateTime @default(now())
}

model Output {
    id      String  @id @default(cuid())
    type    String      // notes, flashcards, quiz, slides
    content Json
    transcriptId String
    transcript Transcript @relation(fields: [transcriptId], references: [id])
    createdAt DateTime    @default(now())
}

```

3. Core Domain Entities

- o Refine existing entities (FileEntity, TranscriptEntity, OutputEntity)

- Add CourseEntity, UserEntity
- Implement value objects (FlashCard, QuizQuestion, Note)

4. Auth0 Integration

- Install `@auth0/nextjs-auth0`
- Configure protected routes
- CreateAuthProvider wrapper

Deliverables:

- `.env.local` configured
 - Prisma schema defined
 - Auth0 working on localhost
 - Basic project structure validated
-

Phase 2: File Ingestion & Storage (Days 3-4)

Goal: Handle file uploads and text extraction

Tasks:

1. UploadThing Integration

```
// src/adapters/uploadthing.adapter.ts
import { createUploadthing } from "uploadthing/next";

const f = createUploadthing();

export const uploadRouter = {
  courseFiles: f({
    pdf: { maxSize: "16MB", maxFileCount: 10 },
    image: { maxSize: "4MB", maxFileCount: 20 },
    video: { maxSize: "256MB", maxFileCount: 5 },
  })
    .middleware(async ({ req }) => {
      const user = await getUser(req);
      return { userId: user.id };
    })
    .onUploadComplete(async ({ metadata, file }) => {
      await saveFileToDatabase(file, metadata.userId);
    }),
};
```

2. Ingest Service

```
// src/services/ingest.service.ts
export class IngestService {
  constructor(
    private supabase: IRepository,
    private textExtractor: ITextExtractor
  ) {}

  async processFile(file: FileEntity): Promise<string> {
    switch (file.type) {
      case 'pdf':
        return this.textExtractor.extractFromPDF(file.url);
      case 'pptx':
        return this.textExtractor.extractFromPPTX(file.url);
      case 'docx':
        return this.textExtractor.extractFromDOCX(file.url);
      default:
        throw new Error(`Unsupported file type: ${file.type}`);
    }
  }
}
```

3. Text Extraction Adapters

- PDF: Use `pdf-parse` or `pdfjs-dist`
- PPTX: Use `officegen` or API call
- DOCX: Use `mammoth`

Deliverables:

- File upload UI component
- `/api/ingest` route handler
- Text extraction working for PDF, PPTX, DOCX
- Files saved to Supabase Storage

Phase 3: ElevenLabs Transcription (Days 5-6)

Goal: Use ElevenLabs Speech-to-Text API for video/audio transcription

Tasks:

1. ElevenLabs Adapter Implementation

```
// src/adapters/elevenlabs.adapter.ts
import { ElevenLabsClient } from "@elevenlabs/elevenlabs-js";

export class ElevenLabsAdapter implements ITranscriber {
  private client: ElevenLabsClient;

  constructor(apiKey: string) {
    this.client = new ElevenLabsClient({ apiKey });
  }

  async transcribe(audioUrl: string): Promise<string> {
    // Download audio file
    const response = await fetch(audioUrl);
    const arrayBuffer = await response.arrayBuffer();
    const audioFile = new File([arrayBuffer], 'audio.mp3', {
      type: response.headers.get('content-type') || 'audio/mpeg'
    });

    // Transcribe using ElevenLabs
    const result = await this.client.speechToText.convert({
      audio: audioFile,
      model_id: "scribe-v1", // ElevenLabs transcription model
    });

    return result.text;
  }

  async transcribeWithTimestamps(audioUrl: string): Promise<{
    text: string;
    segments?: Array<{ start: number; end: number; text: string }>;
  }> {
    const response = await fetch(audioUrl);
    const arrayBuffer = await response.arrayBuffer();
    const audioFile = new File([arrayBuffer], 'audio.mp3', {
      type: response.headers.get('content-type') || 'audio/mpeg'
    });

    const result = await this.client.speechToText.convertWithTimestamps({
      audio: audioFile,
      model_id: "scribe-v1",
    });

    return {
      text: result.text,
      segments: result.segments?.map(s => ({
        start: s.start_time,
        end: s.end_time,
        text: s.text
      }))
    };
  }
}
```

```
    }  
}
```

2. Transcribe Service

```
// src/services/transcribe.service.ts
import { PrismaClient } from '@prisma/client';
import { ITranscriber } from '@/domain/interfaces/ITranscriber';
import { TranscriptEntity } from '@/domain/entities/TranscriptEntity';

export class TranscribeService {
  constructor(
    private transcriber: ITranscriber,
    private prisma: PrismaClient
  ) {}

  async transcribeFile(fileId: string): Promise<TranscriptEntity> {
    // Get file from database
    const file = await this.prisma.file.findUnique({
      where: { id: fileId }
    });

    if (!file) {
      throw new Error(`File not found: ${fileId}`);
    }

    // Check if file is audio/video
    const audioVideoTypes = [
      'audio/mpeg', 'audio/mp3', 'audio/wav', 'audio/m4a',
      'video/mp4', 'video/webm', 'video/quicktime'
    ];

    if (!audioVideoTypes.includes(file.type)) {
      throw new Error(`File type ${file.type} is not audio/video`);
    }

    // Transcribe using ElevenLabs
    const text = await this.transcriber.transcribe(file.url);

    // Save transcript to database
    const transcript = await this.prisma.transcript.create({
      data: {
        content: text,
        fileId: file.id,
      }
    });

    return new TranscriptEntity(
      transcript.id,
      transcript.fileId,
      transcript.content
    );
  }

  async getTranscript(transcriptId: string): Promise<TranscriptEntity | null> {
  }
}
```

```
const transcript = await this.prisma.transcript.findUnique({
  where: { id: transcriptId }
});

if (!transcript) return null;

return new TranscriptEntity(
  transcript.id,
  transcript.fileId,
  transcript.content
);
}
```

3. API Route Handler

```
// src/app/api/transcribe/route.ts
import { NextResponse } from 'next/server';
import { getSession } from '@/lib/auth';
import { TranscribeService } from '@/services/transcribe.service';
import { ElevenLabsAdapter } from '@/adapters/elevenlabs.adapter';
import { PrismaClient } from '@prisma/client';

export async function POST(req: Request) {
  try {
    // Verify authentication
    const session = await getSession();
    if (!session?.user) {
      return NextResponse.json(
        { error: 'Unauthorized' },
        { status: 401 }
      );
    }
  }

  // Parse request
  const { fileId } = await req.json();

  if (!fileId) {
    return NextResponse.json(
      { error: 'Missing fileId' },
      { status: 400 }
    );
  }

  // Initialize services
  const prisma = new PrismaClient();
  const elevenlabs = new ElevenLabsAdapter(
    process.env.ELEVENLABS_API_KEY!
  );
  const transcribeService = new TranscribeService(elevenlabs, prisma);

  // Transcribe file
  const transcript = await transcribeService.transcribeFile(fileId);

  return NextResponse.json({
    success: true,
    transcript: {
      id: transcript.id,
      fileId: transcript.fileId,
      text: transcript.text,
    }
  });
}

} catch (error: any) {
  console.error('Transcription error:', error);
  return NextResponse.json(
```

```
        { error: error.message || 'Transcription failed' },
        { status: 500 }
    );
}
}

export async function GET(req: Request) {
    try {
        const session = await getSession();
        if (!session?.user) {
            return NextResponse.json(
                { error: 'Unauthorized' },
                { status: 401 }
            );
        }
    }

    const { searchParams } = new URL(req.url);
    const transcriptId = searchParams.get('id');

    if (!transcriptId) {
        return NextResponse.json(
            { error: 'Missing transcript id' },
            { status: 400 }
        );
    }

    const prisma = new PrismaClient();
    const elevenlabs = new ElevenLabsAdapter(
        process.env.ELEVENLABS_API_KEY!
    );
    const transcribeService = new TranscribeService(elevenlabs, prisma);

    const transcript = await transcribeService.getTranscript(transcriptId);

    if (!transcript) {
        return NextResponse.json(
            { error: 'Transcript not found' },
            { status: 404 }
        );
    }

    return NextResponse.json({
        success: true,
        transcript: {
            id: transcript.id,
            fileId: transcript.fileId,
            text: transcript.text,
        }
    });
}

} catch (error: any) {
```

```
    console.error('Get transcript error:', error);
    return NextResponse.json(
      { error: error.message || 'Failed to get transcript' },
      { status: 500 }
    );
  }
}
```

4. Custom Hook

```
// src/hooks/useTranscribe.ts
'use client';

import { useMutation, useQuery, useQueryClient } from '@tanstack/react-query';

export function useTranscribe() {
  const queryClient = useQueryClient();

  const transcribeMutation = useMutation({
    mutationFn: async (fileId: string) => {
      const res = await fetch('/api/transcribe', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ fileId }),
      });

      if (!res.ok) {
        const error = await res.json();
        throw new Error(error.error || 'Transcription failed');
      }

      return res.json();
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['transcripts'] });
    },
  });

  return {
    transcribe: transcribeMutation.mutateAsync,
    isTranscribing: transcribeMutation.isPending,
    error: transcribeMutation.error,
  };
}

export function useTranscript(transcriptId: string | null) {
  return useQuery({
    queryKey: ['transcript', transcriptId],
    queryFn: async () => {
      if (!transcriptId) return null;

      const res = await fetch(`api/transcribe?id=${transcriptId}`);
      if (!res.ok) throw new Error('Failed to fetch transcript');
      return res.json();
    },
    enabled: !!transcriptId,
  });
}
```

Deliverables:

- ElevenLabs adapter with Speech-to-Text integration
 - `/api/transcribe` route handler (POST for transcription, GET for retrieval)
 - TranscribeService with Prisma integration
 - Custom React hooks for transcription
 - Video/audio transcription working end-to-end
-

Phase 4: AI Agents (Days 7-10)

Goal: Implement specialized AI agents using Gemini/OpenRouter

Tasks:

1. Model Client Abstraction

```
// src/domain/interfaces/IModelClient.ts
export interface CompletionOptions {
  temperature?: number;
  maxTokens?: number;
  systemPrompt?: string;
}

export interface IModelClient {
  complete(
    prompt: string,
    options?: CompletionOptions
  ): Promise<string>;
}
```

2. Gemini Adapter

```
// src/adapters/gemini.adapter.ts
import { GoogleGenerativeAI } from "@google/generative-ai";

export class GeminiAdapter implements IModelClient {
  private genAI: GoogleGenerativeAI;

  constructor(apiKey: string) {
    this.genAI = new GoogleGenerativeAI(apiKey);
  }

  async complete(prompt: string, options?: CompletionOptions): Promise<string> {
    const model = this.genAI.getGenerativeModel({
      model: "gemini-1.5-pro",
      systemInstruction: options?.systemPrompt,
    });

    const result = await model.generateContent(prompt);
    return result.response.text();
  }
}
```

3. Agent Implementations

```

// src/services/agents/NotesAgent.ts
export class NotesAgent implements IAgent {
  name = "Notes Agent";

  constructor(private modelClient: IModelClient) {}

  async process({ transcript }: { transcript: string }): Promise<string> {
    const prompt = `
      You are an expert note-taker. Transform the following lecture transcript
      into structured, comprehensive study notes using markdown format.

      Include:
      - Key concepts and definitions
      - Important examples
      - Summary sections

      Transcript:
      ${input.transcript}
    `;

    return this.modelClient.complete(prompt, {
      systemPrompt: "You are a study assistant creating organized notes.",
      temperature: 0.3,
    });
  }
}

// Similar implementations for FlashcardAgent, QuizAgent, SlidesAgent

```

4. Generate Service Orchestration

```

// src/services/generate.service.ts
export class GenerateService {
  private agents: Map<string, IAgent>;

  constructor(modelClient: IModelClient) {
    this.agents = new Map([
      ['notes', new NotesAgent(modelClient)],
      ['flashcards', new FlashcardAgent(modelClient)],
      ['quiz', new QuizAgent(modelClient)],
      ['slides', new SlidesAgent(modelClient)],
    ]);
  }

  async generate(
    transcriptId: string,
    type: AgentType
  ): Promise<OutputEntity> {
    const transcript = await this.repository.getTranscript(transcriptId);
    const agent = this.agents.get(type);

    if (!agent) throw new Error(`Unknown agent type: ${type}`);

    const content = await agent.process({ transcript: transcript.content });
    return this.repository.saveOutput(transcriptId, type, content);
  }

  async generateAll(transcriptId: string): Promise<OutputEntity[]> {
    const results = await Promise.allSettled(
      Array.from(this.agents.keys()).map(type =>
        this.generate(transcriptId, type as AgentType)
      )
    );

    return results
      .filter(r => r.status === 'fulfilled')
      .map(r => (r as PromiseFulfilledResult<OutputEntity>).value);
  }
}

```

5. Prompt Engineering

```
// src/domain/types/prompts.ts
export const AGENT_PROMPTS = {
  notes: {
    system: "You are an expert educator creating study notes.",
    template: (transcript: string) => `
      Create comprehensive study notes from this lecture transcript.
      Format using markdown with clear headings and bullet points.

      ${transcript}
    `,
  },
  flashcards: {
    system: "You are a flashcard creator following spaced repetition principles.",
    template: (transcript: string) => `
      Generate 15-20 flashcards from this content.
      Return as JSON array: [{"front": "question", "back": "answer"}]

      ${transcript}
    `,
  },
  quiz: {
    system: "You are a quiz creator making engaging assessments.",
    template: (transcript: string, difficulty: string) => `
      Create a ${difficulty} quiz with 10 multiple-choice questions.
      Return as JSON: {

        "questions": [
          {
            "question": "text",
            "options": ["A", "B", "C", "D"],
            "correct": 0
          }
        ]
      }

      ${transcript}
    `,
  },
  slides: {
    system: "You are a presentation designer creating educational slides.",
    template: (transcript: string) => `
      Extract 8-12 key points for presentation slides.
      Return as JSON array: [
        {
          "title": "slide title",
          "bullets": ["point 1", "point 2"]
        }
      ]

      ${transcript}
    `,
  },
}
```

```
 },  
};
```

Deliverables:

- All 4 agents implemented (Notes, Flashcards, Quiz, Slides)
 - `/api/generate` route handler
 - Agent chat interface for refinement
 - Fallback to OpenRouter if Gemini fails
-

Phase 5: Canvas Integration (Days 11-12)

Goal: Sync course files from Canvas LMS

Tasks:

1. Canvas Adapter

```

// src/adapters/canvas.adapter.ts
export class CanvasAdapter {
    constructor(private baseUrl: string) {}

    async getCourses(accessToken: string): Promise<Course[]> {
        const response = await fetch(`.${this.baseUrl}/courses`, {
            headers: { Authorization: `Bearer ${accessToken}` },
        });
        return response.json();
    }

    async getCourseFiles(
        courseId: string,
        accessToken: string
    ): Promise<CanvasFile[]> {
        const response = await fetch(
            `.${this.baseUrl}/courses/${courseId}/files`,
            { headers: { Authorization: `Bearer ${accessToken}` } }
        );
        return response.json();
    }

    async downloadFile(fileUrl: string, accessToken: string): Promise<Blob> {
        const response = await fetch(fileUrl, {
            headers: { Authorization: `Bearer ${accessToken}` },
        });
        return response.blob();
    }
}

```

2. Canvas Service

```

// src/services/canvas.service.ts
export class CanvasService {
  constructor(
    private canvasAdapter: CanvasAdapter,
    private ingestService: IngestService,
    private repository: IRepository
  ) {}

  async syncCourses(userId: string, accessToken: string): Promise<void> {
    const courses = await this.canvasAdapter.getCourses(accessToken);
    const eightMonthsAgo = new Date();
    eightMonthsAgo.setMonth(eightMonthsAgo.getMonth() - 8);

    for (const course of courses) {
      if (new Date(course.created_at) < eightMonthsAgo) continue;

      const files = await this.canvasAdapter.getCourseFiles(
        course.id,
        accessToken
      );

      for (const file of files) {
        await this.downloadAndIngest(userId, course.id, file, accessToken);
      }
    }
  }
}

```

3. Canvas UI Flow

- o Token input modal
- o Course selection
- o File sync progress
- o Auto-generate on import

Deliverables:

- /api/canvas-sync route handler
- Canvas integration UI
- Secure token storage (encrypted in Supabase)

Phase 6: Export Pipeline (Days 13-14)

Goal: Generate downloadable study materials

Tasks:

1. Export Service Architecture

```
// src/domain/interfaces/IExporter.ts
export interface IExporter {
  export(output: OutputEntity): Promise<Buffer>;
  getMimeType(): string;
  getFileExtension(): string;
}
```

2. Exporter Implementations

```

// src/services/exporters/PDFExporter.ts
import { jsPDF } from "jspdf";

export class PDFExporter implements IExporter {
  getMimeType() { return 'application/pdf'; }
  getFileExtension() { return '.pdf'; }

  async export(output: OutputEntity): Promise<Buffer> {
    const doc = new jsPDF();
    doc.setFont("helvetica");
    doc.setFontSize(12);
    doc.text(output.content as string, 10, 10);
    return Buffer.from(doc.output('arraybuffer'));
  }
}

// src/services/exporters/AnkiExporter.ts
import genanki from 'genanki';

export class AnkiExporter implements IExporter {
  getMimeType() { return 'application/apkg'; }
  getFileExtension() { return '.apkg'; }

  async export(output: OutputEntity): Promise<Buffer> {
    const deck = new genanki.Deck(Date.now(), 'EduFlow Flashcards');
    const model = new genanki.Model({
      name: 'Simple',
      id: Date.now(),
      flds: [{ name: 'Front' }, { name: 'Back' }],
      tmpls: [
        {
          name: 'Card 1',
          qfmt: '{{Front}}',
          afmt: '{{FrontSide}}<hr>{{Back}}',
        },
      ],
    });

    const cards = output.content as Array<{ front: string; back: string }>;
    cards.forEach(card => {
      deck.addNote(new genanki.Note(model, [card.front, card.back]));
    });

    return genanki.Package(deck).writeToFile();
  }
}

// Similar for CSVExporter, PPTXExporter

```

3. Export Service

```

// src/services/export.service.ts
export class ExportService {
  private exporters: Map<string, IExporter>;

  constructor() {
    this.exporters = new Map([
      ['pdf', new PDFExporter()],
      ['anki', new AnkiExporter()],
      ['csv', new CSVExporter()],
      ['pptx', new PPTXExporter()],
    ]);
  }

  async export(outputId: string, format: ExportFormat): Promise<{
    buffer: Buffer;
    mimeType: string;
    filename: string;
  }> {
    const output = await this.repository.getOutput(outputId);
    const exporter = this.exporters.get(format);

    if (!exporter) throw new Error(`Unsupported format: ${format}`);

    const buffer = await exporter.export(output);
    return {
      buffer,
      mimeType: exporter.getMimeType(),
      filename:
        `eduflow-${output.type}-${Date.now()}${exporter.getFileExtension()}`,
    };
  }
}

```

4. Export API Route

```

// src/app/api/export/route.ts
export async function POST(req: Request) {
  const { outputId, format } = await req.json();

  const exportService = new ExportService();
  const { buffer, mimeType, filename } = await exportService.export(
    outputId,
    format
  );

  return new Response(buffer, {
    headers: {
      'Content-Type': mimeType,
      'Content-Disposition': `attachment; filename="${filename}"`,
    },
  });
}

```

Deliverables:

- PDF export for notes
 - Anki deck (.apkg) export for flashcards
 - CSV export for quizzes
 - PPTX export for slides
 - Export UI buttons
-

Phase 7: Frontend & Canvas UI (Days 15-18)

Goal: Build beautiful, intuitive UI with interactive canvas

Tasks:

1. Dependencies

```

npm install @xyflow/react framer-motion lucide-react
npm install @radix-ui/react-dialog @radix-ui/react-dropdown-menu
npm install react-query @tanstack/react-query
npm install zustand # state management

```

2. Canvas View with React Flow

```

// src/components/CanvasView.tsx
'use client';

import { ReactFlow, Node, Edge, Background } from '@xyflow/react';
import { AgentNode } from './nodes/AgentNode';
import { FileNode } from './nodes/FileNode';
import { OutputNode } from './nodes/OutputNode';

const nodeTypes = {
  agent: AgentNode,
  file: FileNode,
  output: OutputNode,
};

export function CanvasView({ courseId }: { courseId: string }) {
  const [nodes, setNodes] = useState<Node[]>([]);
  const [edges, setEdges] = useState<Edge[]>([]);

  return (
    <div className="w-full h-screen">
      <ReactFlow
        nodes={nodes}
        edges={edges}
        nodeTypes={nodeTypes}
        fitView
      >
        <Background />
      </ReactFlow>
    </div>
  );
}

```

3. Agent Node Component

```
// src/components/nodes/AgentNode.tsx
import { Handle, Position } from '@xyflow/react';
import { motion } from 'framer-motion';

export function AgentNode({ data }: { data: AgentNodeData }) {
  return (
    <motion.div
      initial={{ scale: 0 }}
      animate={{ scale: 1 }}
      className="bg-white rounded-lg shadow-lg p-4 border-2 border-blue-500"
    >
      <Handle type="target" position={Position.Left} />
      <div className="flex items-center gap-2">
        <data.icon className="w-5 h-5" />
        <span className="font-semibold">{data.label}</span>
      </div>
      {data.status === 'processing' && (
        <div className="mt-2 w-full bg-gray-200 rounded-full h-2">
          <div className="bg-blue-500 h-2 rounded-full animate-pulse" />
        </div>
      )}
      <Handle type="source" position={Position.Right} />
    </motion.div>
  );
}
```

4. Agent Chat Interface

```

// src/components/AgentChat.tsx
import { useAgentChat } from '@/hooks/useAgentChat';

export function AgentChat({ agentType, outputId }: Props) {
  const { messages, sendMessage, isLoading } = useAgentChat(agentType, outputId);

  return (
    <div className="flex flex-col h-full">
      <div className="flex-1 overflow-y-auto p-4 space-y-4">
        {messages.map((msg, i) => (
          <div key={i} className={`flex ${msg.role === 'user' ? 'justify-end' : 'justify-start'}`}>
            <div className={`${rounded-lg p-3 max-w-md ${
              msg.role === 'user' ? 'bg-blue-500 text-white' : 'bg-gray-100'
            }`}>
              {msg.content}
            </div>
          </div>
        )));
      </div>
      <form onSubmit={(e) => {
        e.preventDefault();
        sendMessage(e.currentTarget.message.value);
      }} className="p-4 border-t">
        <input
          name="message"
          placeholder="@Notes_Agent make this more concise..."
          className="w-full px-4 py-2 border rounded-lg"
        />
      </form>
    </div>
  );
}

```

5. Dashboard

```

// src/app/dashboard/page.tsx
export default function Dashboard() {
  return (
    <div className="min-h-screen bg-gradient-to-br from-indigo-50 to-white">
      <header className="border-b bg-white/80 backdrop-blur">
        <div className="container mx-auto px-4 py-4 flex items-center justify-between">
          <h1 className="text-2xl font-bold">EduFlow AI</h1>
          <UserNav />
        </div>
      </header>

      <main className="container mx-auto px-4 py-8">
        <div className="grid grid-cols-3 gap-6">
          <CourseGrid />
          <UploadZone />
          <CanvasButton />
        </div>
      </main>
    </div>
  );
}

```

Deliverables:

- Infinite canvas UI with React Flow
- Agent nodes with status indicators
- File upload drag-and-drop zone
- Agent chat interface
- Responsive dashboard
- Beautiful animations with Framer Motion

Phase 8: Testing & Polish (Days 19-21)

Goal: Ensure quality, write tests, optimize performance

Tasks:

1. Unit Tests

```

// __tests__/services/generate.service.test.ts
import { GenerateService } from '@/services/generate.service';
import { MockModelClient } from '@/tests/mocks';

describe('GenerateService', () => {
  it('should generate notes from transcript', async () => {
    const mockClient = new MockModelClient();
    const service = new GenerateService(mockClient);

    const result = await service.generate('transcript-123', 'notes');

    expect(result.type).toBe('notes');
    expect(result.content).toContain('Key Concepts');
  });
});

```

2. Integration Tests

- Test full flow: upload → transcribe → generate → export
- Test Canvas sync
- Test agent refinement chat

3. Performance Optimization

- Lazy load agent nodes
- Implement React Query caching
- Add loading skeletons
- Optimize Gemini API calls (batching)

4. Error Handling

- Toast notifications for errors
- Retry logic for API failures
- Graceful degradation

5. Documentation

- Update README with setup instructions
- Add JSDoc comments
- Create API documentation
- Record demo video

Deliverables:

- 80%+ test coverage
 - Error handling throughout
 - Performance optimized
 - Complete documentation
-

Dependencies & Setup

Install Command

```
npm install --save \
@auth0/nextjs-auth0 \
@google/generative-ai \
@prisma/client \
@supabase/supabase-js \
@tanstack/react-query \
@xyflow/react \
framer-motion \
lucide-react \
uploadthing \
@uploadthing/react \
zustand \
jspdf \
genanki \
pdf-parse \
mammoth \
axios

npm install --save-dev \
prisma \
@types/node \
@types/react \
@types/react-dom \
vitest \
@testing-library/react \
@testing-library/jest-dom
```

Prisma Setup

```
npx prisma init
npx prisma db push
npx prisma generate
```

Auth0 Setup

```
openssl rand -hex 32 # Generate AUTH0_SECRET
```

💡 API Integration Strategy

1. Gemini API (Primary AI)

- **Use Case:** All agent generations
- **Rate Limit:** 60 requests/minute
- **Fallback:** OpenRouter

2. OpenRouter (Fallback AI)

- **Use Case:** When Gemini fails/rate-limited
- **Models:** Claude 3.5, GPT-4
- **Cost:** Pay-per-token

3. Whisper (Self-Hosted)

- **Use Case:** Video transcription
- **Deployment:** Digital Ocean droplet
- **Model:** `whisper-small` (fast, accurate enough)

4. UploadThing

- **Use Case:** File uploads
- **Limit:** 2GB per upload
- **Storage:** Automatic CDN

5. Supabase

- **Use Case:** Database + file storage
- **Features:** Row-level security, real-time subscriptions

6. ElevenLabs (Future)

- **Use Case:** Text-to-speech for notes

- Priority: Low (add after MVP)
-

⌚ Frontend Architecture

State Management

```
// src/store/useCanvasStore.ts
import { create } from 'zustand';

interface CanvasStore {
  nodes: Node[];
  edges: Edge[];
  addNode: (node: Node) => void;
  addEdge: (edge: Edge) => void;
  updateNode: (id: string, data: Partial<Node>) => void;
}

export const useCanvasStore = create<CanvasStore>((set) => ({
  nodes: [],
  edges: [],
  addNode: (node) => set((state) => ({ nodes: [...state.nodes, node] })),
  addEdge: (edge) => set((state) => ({ edges: [...state.edges, edge] })),
  updateNode: (id, data) => set((state) => ({
    nodes: state.nodes.map(n => n.id === id ? { ...n, ...data } : n),
  })),
}));
```

React Query Setup

```
// src/providers/QueryProvider.tsx
'use client';

import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import { useState } from 'react';

export function QueryProvider({ children }: { children: React.ReactNode }) {
  const [queryClient] = useState(() => new QueryClient({
    defaultOptions: {
      queries: {
        staleTime: 60 * 1000, // 1 minute
        refetchOnWindowFocus: false,
      },
    },
  }));
}

return (
  <QueryClientProvider client={queryClient}>
    {children}
  </QueryClientProvider>
);
}
```

Custom Hooks

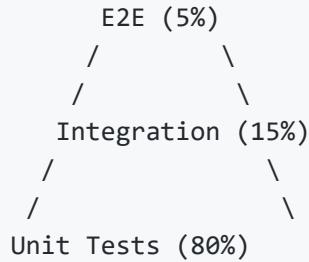
```
// src/hooks/useGenerate.ts
import { useMutation, useQueryClient } from '@tanstack/react-query';

export function useGenerate() {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async ({ transcriptId, type }: GenerateInput) => {
      const res = await fetch('/api/generate', {
        method: 'POST',
        body: JSON.stringify({ transcriptId, type }),
      });
      return res.json();
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['outputs'] });
    },
  });
}
```

✍️ Testing Strategy

Test Pyramid



Tools

- **Unit:** Vitest
- **Integration:** Vitest + MSW (Mock Service Worker)
- **E2E:** Playwright
- **Coverage:** Vitest Coverage

Example Test Suite

```
// __tests__/agents/NotesAgent.test.ts
import { describe, it, expect, vi } from 'vitest';
import { NotesAgent } from '@/services/agents/NotesAgent';

describe('NotesAgent', () => {
  it('should generate structured notes', async () => {
    const mockClient = {
      complete: vi.fn().mockResolvedValue('# Key Concepts\n- Point 1'),
    };

    const agent = new NotesAgent(mockClient);
    const result = await agent.process({ transcript: 'Lorem ipsum...' });

    expect(result).toContain('# Key Concepts');
    expect(mockClient.complete).toHaveBeenCalledWith(
      expect.stringContaining('Lorem ipsum'),
      expect.any(Object)
    );
  });
});
```

Deployment Checklist

Vercel Deployment

```
# Install Vercel CLI  
npm i -g vercel  
  
# Deploy  
vercel --prod
```

Environment Variables (Vercel)

Add all `.env.local` variables to Vercel dashboard.

Database Migrations

```
npx prisma migrate deploy
```

Whisper Server (Digital Ocean)

```
# Create systemd service for auto-restart  
sudo systemctl enable whisper-api  
sudo systemctl start whisper-api
```

Success Metrics

Technical Excellence

- 80%+ test coverage
- All SOLID principles demonstrated
- Zero TypeScript errors
- Lighthouse score > 90

Feature Completeness

- All 4 agents working
- Canvas integration functional

- All export formats working
- Beautiful, responsive UI

Demo Quality

- 3-minute demo video
 - Live deployed version
 - Clean GitHub repo
 - Professional README
-

⌚ Next Steps

1. Run setup commands:

```
npm install
npx prisma db push
npm run dev
```

2. Set up Digital Ocean droplet for Whisper

3. Start with Phase 1: Auth0 + Prisma + basic UI

4. Iterate through phases following this plan

5. Test continuously as you build

📞 Questions to Answer Before Starting

1. Do you have access to the Digital Ocean droplet?
 2. Should we implement ElevenLabs now or later?
 3. What's the priority: feature completeness vs. polish?
 4. Do you have a Canvas account to test integration?
 5. What's the hackathon deadline?
-

Ready to build? Let's start with Phase 1! 