# Educational Study Material Platform - Detailed Plan

## Project Overview

A multi-institutional platform where students can access curated study materials (notes, PPTs, PDFs) with AI-powered summaries and search capabilities.

---

## 1. SYSTEM ARCHITECTURE

### High-Level Architecture


### Technology Stack Details

**Frontend:**

- **Web:** React.js 18+ with TypeScript
  - React Router for navigation
  - Tailwind CSS for styling
  - Axios for API calls

    - React Query for state management
- **Mobile:** React Native (Phase 2)
    - Expo for easier development
    - Same core components as web

**Backend:**

- **Runtime:** Node.js 18+ LTS
- **Framework:** Express.js
- **Language:** TypeScript
- **API Style:** RESTful

**Database:**

- **Primary:** PostgreSQL 15+
    - Structured relational data
    - ACID compliance
    - Full-text search capability

**File Storage:**

- **Service:** AWS S3 or Cloudflare R2
- **CDN:** CloudFront for faster delivery

**AI Integration:**

- **Provider:** Anthropic Claude API
- **Use Cases:**
    - Document summarization
    - Q&A (Phase 3)

**Authentication:**

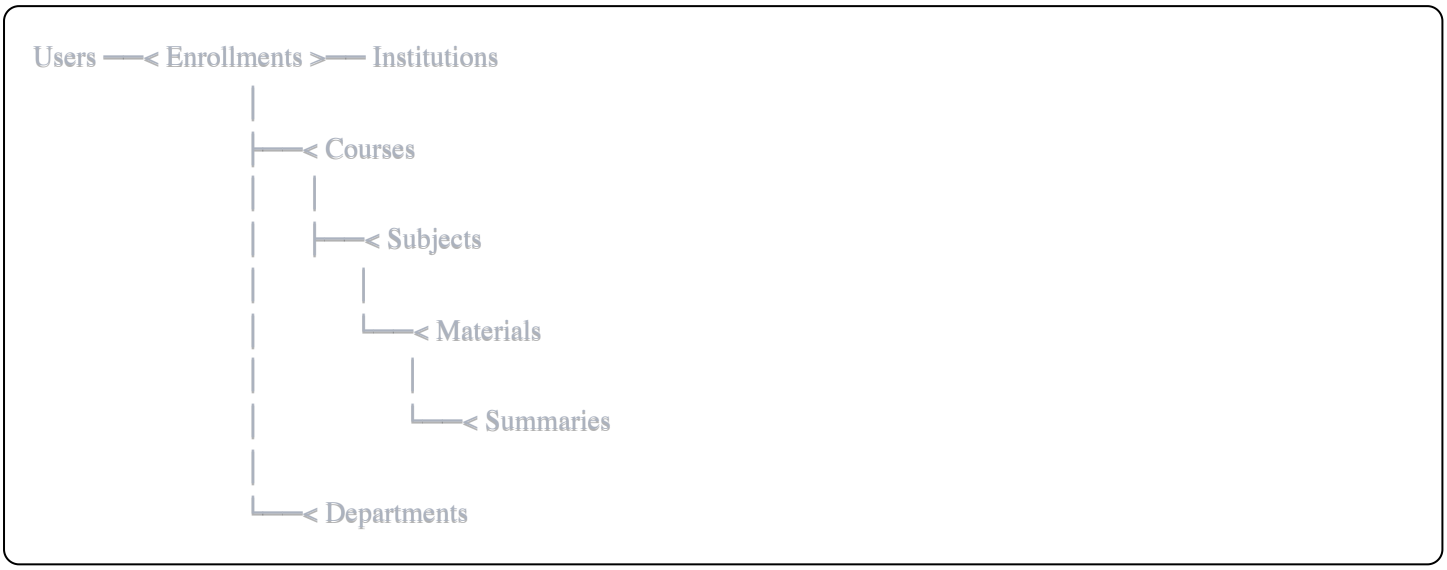
- **Method:** JWT (JSON Web Tokens)
- **Storage:** HTTP-only cookies
- **Password:** bcrypt hashing

**Deployment:**

- **Frontend:** Vercel or Netlify
- **Backend:** Railway, Render, or AWS EC2
- **Database:** Railway, Supabase, or AWS RDS

## 2. DATABASE SCHEMA

### Entity Relationship Diagram




### Table Structures

#### users

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(255) NOT NULL,
    role VARCHAR(50) DEFAULT 'student', -- student, admin, super_admin
    is_verified BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP
);
```

#### institutions

```sql
```

```sql
CREATE TABLE institutions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    code VARCHAR(50) UNIQUE NOT NULL, -- e.g., "MIT", "STANFORD"
    location VARCHAR(255),
    type VARCHAR(50), -- university, college, school
    logo_url TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT true
);
```

### departments

sql

```sql
CREATE TABLE departments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    institution_id UUID REFERENCES institutions(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL, -- Computer Science, Mechanical, etc.
    code VARCHAR(50) NOT NULL, -- CS, ME, EE
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(institution_id, code)
);
```

### courses

sql

```sql
CREATE TABLE courses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    department_id UUID REFERENCES departments(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL, -- B.Tech, M.Tech, etc.
    code VARCHAR(50) NOT NULL,
    duration_years INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(department_id, code)
);
```

### subjects

sql

```sql
CREATE TABLE subjects (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    course_id UUID REFERENCES courses(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL, -- Data Structures, DBMS, etc.
    code VARCHAR(50) NOT NULL, -- CS201, CS301
    semester INTEGER, -- 1-8
    year INTEGER, -- 1-4
    credits INTEGER,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(course_id, code)
);
```

### materials

```sql
CREATE TABLE materials (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    subject_id UUID REFERENCES subjects(id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    type VARCHAR(50) NOT NULL, -- pdf, ppt, doc, notes
    file_url TEXT NOT NULL, -- S3 URL
    file_size BIGINT, -- in bytes
    file_name VARCHAR(255),
    uploaded_by UUID REFERENCES users(id),
    upload_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_processed BOOLEAN DEFAULT false, -- AI summary generated?
    view_count INTEGER DEFAULT 0,
    download_count INTEGER DEFAULT 0,
    tags TEXT[], -- array of tags
    is_active BOOLEAN DEFAULT true
);
```

### ai_summaries

```sql
```

```sql
CREATE TABLE ai_summaries (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    material_id UUID REFERENCES materials(id) ON DELETE CASCADE,
    summary_text TEXT NOT NULL,
    key_points TEXT[], -- array of key points
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    model_used VARCHAR(100), -- claude-3-sonnet-20240229
    processing_time INTEGER, -- in seconds
    UNIQUE(material_id)
);
```

### enrollments (user-institution relationship)

```sql
CREATE TABLE enrollments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    institution_id UUID REFERENCES institutions(id) ON DELETE CASCADE,
    course_id UUID REFERENCES courses(id),
    semester INTEGER,
    year INTEGER,
    enrollment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT true,
    UNIQUE(user_id, institution_id)
);
```

### search_logs (for analytics)

```sql
CREATE TABLE search_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    search_query TEXT,
    filters JSONB, -- store filter criteria
    results_count INTEGER,
    searched_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

## 3. API ENDPOINTS DESIGN

**Base URL:** `https://api.studyplatform.com/v1`

### Authentication Endpoints

#### POST /auth/register

**Request:**

```
json

{
  "email": "student@example.com",
  "password": "securePassword123",
  "fullName": "John Doe",
  "institutionId": "uuid"
}
```

**Response:**

```
json

{
  "success": true,
  "message": "Registration successful. Please verify your email.",
  "userId": "uuid"
}
```

#### POST /auth/login

**Request:**

```
json

{
  "email": "student@example.com",
  "password": "securePassword123"
}
```

**Response:**

```
json
```

```
{
  "success": true,
  "token": "jwt_token_here",
  "user": {
    "id": "uuid",
    "email": "student@example.com",
    "fullName": "John Doe",
    "role": "student"
  }
}
```

**POST /auth/logout**

**Headers:** `Authorization: Bearer {token}` **Response:**

```
json

{
  "success": true,
  "message": "Logged out successfully"
}
```

**Institution Endpoints**

**GET /institutions**

**Query Params:** `?page=1&limit=20&search=MIT` **Response:**

```
json

{
  "success": true,
  "data": [
    {
      "id": "uuid",
      "name": "Massachusetts Institute of Technology",
      "code": "MIT",
      "location": "Cambridge, MA",
      "logoUrl": "https://..."
    }
  ],
  "pagination": {
    "total": 100,
    "page": 1,
    "limit": 20
  }
}
```

### GET /institutions/:id

### Response:

```json
{
  "success": true,
  "data": {
    "id": "uuid",
    "name": "MIT",
    "departments": [...],
    "courses": [...]
  }
}
```

### Course Hierarchy Endpoints

### GET /institutions/:institutionId/departments

### GET /departments/:departmentId/courses

### GET /courses/:courseId/subjects

### Materials Endpoints

### GET /materials

### Query Params:

- ?subjectId=uuid
- &type=pdf,ppt
- &semester=3
- &search=data structures
- &page=1&limit=20

### Response:

```json
```

```
{
  "success": true,
  "data": [
    {
      "id": "uuid",
      "title": "Introduction to Data Structures",
      "description": "Comprehensive notes on...",
      "type": "pdf",
      "fileUrl": "https://...",
      "fileSize": 2048576,
      "uploadDate": "2024-01-15T10:30:00Z",
      "viewCount": 150,
      "downloadCount": 45,
      "tags": ["arrays", "linked-lists"],
      "hasSummary": true
    }
  ],
  "pagination": {...}
}
```


### GET /materials/:id

### Response:


```
json

{
  "success": true,
  "data": {
    "id": "uuid",
    "title": "...",
    "fileUrl": "...",
    "summary": {
      "summaryText": "This document covers...",
      "keyPoints": [
        "Arrays are contiguous memory structures",
        "Linked lists use dynamic memory allocation"
      ]
    },
    "subject": {
      "name": "Data Structures",
      "code": "CS201"
    }
  }
}
```

**POST /materials (Admin only)**

**Headers:** Authorization: Bearer {admin_token} **Request:** (multipart/form-data)

```
file: [PDF/PPT file]
title: "Introduction to Algorithms"
description: "..."
subjectId: "uuid"
type: "pdf"
tags: ["algorithms", "sorting"]
```

**Response:**

```json
{
  "success": true,
  "message": "Material uploaded successfully",
  "materialId": "uuid",
  "processingStatus": "queued" // AI summary generation queued
}
```

**DELETE /materials/:id (Admin only)**

**AI Summary Endpoints**

**POST /materials/:id/generate-summary (Admin only)**

Trigger AI summary generation for a material **Response:**

```json
{
  "success": true,
  "message": "Summary generation started",
  "estimatedTime": 30 // seconds
}
```

**GET /materials/:id/summary**

**Response:**

```json
```

```
{
  "success": true,
  "data": {
    "summaryText": "...",
    "keyPoints": [...],
    "generatedAt": "2024-01-15T10:30:00Z"
  }
}
```

### Search Endpoint

### GET /search

**Query Params:** `?q=algorithm&institutionId=uuid&type=pdf&semester=3` **Response:**

```
json

{
  "success": true,
  "query": "algorithm",
  "results": [
    {
      "type": "material",
      "relevance": 0.95,
      "data": {...}
    }
  ],
  "count": 25
}
```

### User Profile Endpoints

### GET /users/me

**Headers:** `Authorization: Bearer {token}` **Response:**

```
json

{
  "success": true,
  "data": {
    "id": "uuid",
    "email": "...",
    "fullName": "...",
    "enrollments": [...]
  }
}
```

**PUT /users/me**

Update user profile

---

## 4. FRONTEND STRUCTURE

### Project Structure

```
study-platform-frontend/
├── public/
├── src/
│   ├── components/
│   │   ├── auth/
│   │   │   ├── LoginForm.tsx
│   │   │   ├── RegisterForm.tsx
│   │   │   └── ProtectedRoute.tsx
│   │   ├── common/
│   │   │   ├── Header.tsx
│   │   │   ├── Footer.tsx
│   │   │   ├── Sidebar.tsx
│   │   │   ├── SearchBar.tsx
│   │   │   └── LoadingSpinner.tsx
│   │   ├── materials/
│   │   │   ├── MaterialCard.tsx
│   │   │   ├── MaterialList.tsx
│   │   │   ├── MaterialDetail.tsx
│   │   │   ├── MaterialUpload.tsx (admin)
│   │   │   └── SummaryDisplay.tsx
│   │   ├── filters/
│   │   │   ├── FilterPanel.tsx
│   │   │   ├── InstitutionFilter.tsx
│   │   │   ├── SemesterFilter.tsx
│   │   │   └── TypeFilter.tsx
│   │   └── admin/
│   │       ├── Dashboard.tsx
│   │       ├── MaterialManager.tsx
│   │       └── InstitutionManager.tsx
│   ├── pages/
│   │   ├── Home.tsx
│   │   ├── Login.tsx
│   │   ├── Register.tsx
│   │   ├── Browse.tsx
│   │   ├── MaterialView.tsx
│   │   ├── Search.tsx
│   │   ├── Profile.tsx
```

```
│   │   └── admin/
│   │       └── AdminDashboard.tsx
│   ├── services/
│   │   ├── api.ts (axios instance)
│   │   ├── authService.ts
│   │   ├── materialService.ts
│   │   ├── institutionService.ts
│   │   └── searchService.ts
│   ├── hooks/
│   │   ├── useAuth.ts
│   │   ├── useMaterials.ts
│   │   ├── useSearch.ts
│   │   └── useFilters.ts
│   ├── context/
│   │   ├── AuthContext.tsx
│   │   └── ThemeContext.tsx
│   ├── utils/
│   │   ├── constants.ts
│   │   ├── helpers.ts
│   │   └── validators.ts
│   ├── types/
│   │   ├── auth.types.ts
│   │   ├── material.types.ts
│   │   └── api.types.ts
│   ├── App.tsx
│   └── index.tsx
├── package.json
└── tsconfig.json
```

**Key Pages & Features**

**1. Home Page**

- Welcome banner
- Institution selector
- Quick search
- Featured materials
- Recent uploads

**2. Browse Page**

- Hierarchical navigation: Institution → Department → Course → Subject
- Material cards with thumbnails
- Filter sidebar (type, semester, year)

- Sort options (newest, most viewed, most downloaded)
- Pagination

**3. Material Detail Page**

- File viewer (PDF/PPT preview)
- Download button
- AI Summary section (collapsible)
- Key points list
- Related materials
- Metadata (upload date, views, downloads)

**4. Search Page**

- Advanced search form
- Real-time suggestions
- Filter options
- Results grouped by type
- Search history

**5. Admin Dashboard**

- Upload materials
- Manage institutions/courses/subjects
- View analytics (most viewed, download stats)
- Generate AI summaries
- User management

---

## 5. BACKEND STRUCTURE

**Project Structure**

```
study-platform-backend/
├── src/
│   ├── config/
│   │   ├── database.ts
│   │   ├── aws.ts
│   │   ├── claude.ts
│   │   └── env.ts
│   ├── controllers/
```

```
│   │   ├── authController.ts
│   │   ├── institutionController.ts
│   │   ├── materialController.ts
│   │   ├── summaryController.ts
│   │   └── searchController.ts
│   ├── models/
│   │   ├── User.ts
│   │   ├── Institution.ts
│   │   ├── Material.ts
│   │   └── Summary.ts
│   ├── routes/
│   │   ├── authRoutes.ts
│   │   ├── institutionRoutes.ts
│   │   ├── materialRoutes.ts
│   │   ├── summaryRoutes.ts
│   │   └── index.ts
│   ├── middleware/
│   │   ├── auth.ts (JWT verification)
│   │   ├── adminAuth.ts
│   │   ├── validation.ts
│   │   ├── errorHandler.ts
│   │   ├── rateLimit.ts
│   │   └── upload.ts (multer)
│   ├── services/
│   │   ├── authService.ts
│   │   ├── fileService.ts (S3 operations)
│   │   ├── aiService.ts (Claude integration)
│   │   ├── searchService.ts
│   │   └── emailService.ts
│   ├── utils/
│   │   ├── logger.ts
│   │   ├── validators.ts
│   │   └── helpers.ts
│   ├── types/
│   │   └── express.d.ts
│   ├── app.ts
│   └── server.ts
├── prisma/ (or migrations/)
│   ├── schema.prisma
│   └── migrations/
├── tests/
│   ├── unit/
│   └── integration/
├── package.json
├── tsconfig.json
└── .env.example
```

### Key Services

### AI Service (aiService.ts)

```typescript
class AIService {
  async generateSummary(fileUrl: string, fileType: string) {
    // 1. Download file from S3
    // 2. Extract text (use pdf-parse for PDFs)
    // 3. Call Claude API with prompt
    // 4. Return structured summary
  }

  async answerQuestion(materialId: string, question: string) {
    // For Phase 3 - chatbot functionality
  }
}
```

### File Service (fileService.ts)

```typescript
class FileService {
  async uploadToS3(file: File, path: string) {
    // Upload file to S3
    // Return public URL
  }

  async deleteFromS3(fileUrl: string) {
    // Delete file from S3
  }

  async getSignedUrl(fileUrl: string) {
    // Generate temporary download link
  }
}
```

---

## 6. AI INTEGRATION DETAILS

### Document Processing Flow

```
1. Admin uploads file
   ↓
2. File stored in S3
```

```
  ↓
3. Material record created in DB
  ↓
4. Background job triggered
  ↓
5. Extract text from file
  ↓
6. Call Claude API with text
  ↓
7. Parse and structure response
  ↓
8. Save summary to DB
  ↓
9. Update material.is_processed = true
```

### Claude API Integration

### Summary Generation Prompt

typescript

```
const prompt = `You are an educational content summarizer.
Analyze the following study material and provide:

1. A concise summary (150-200 words)
2. 5-7 key points or concepts
3. Main topics covered

Material Content:
${documentText}

Respond in JSON format:
{
  "summary": "...",
  "keyPoints": ["...", "..."],
  "topics": ["...", "..."]
}`;
```

### API Call Example

typescript

```
import Anthropic from '@anthropic-ai/sdk';

const client = new Anthropic({
  apiKey: process.env.CLAUDE_API_KEY,
});

const message = await client.messages.create({
  model: 'claude-3-sonnet-20240229',
  max_tokens: 1024,
  messages: [{
    role: 'user',
    content: prompt
  }]
});

const summary = JSON.parse(message.content[0].text);
```

**Text Extraction Libraries**

**For PDFs:**

```
typescript

import pdf from 'pdf-parse';

const dataBuffer = await downloadFromS3(fileUrl);
const data = await pdf(dataBuffer);
const text = data.text;
```

**For PPT:**

```
typescript

import officeParser from 'officeparser';

const text = await officeParser.parseOfficeAsync(filePath);
```

---

## 7. DEVELOPMENT PHASES

### Phase 1: Foundation (Weeks 1-2)

**Goal:** Basic infrastructure and authentication

**Tasks:**

- [ ] Set up Git repository

- [ ] Initialize React project with TypeScript
- [ ] Initialize Node.js backend with Express
- [ ] Set up PostgreSQL database
- [ ] Create database schema and migrations
- [ ] Implement user registration
- [ ] Implement login with JWT
- [ ] Create protected routes
- [ ] Basic frontend routing
- [ ] Set up environment variables

**Deliverables:**

- Working authentication system
- Database with initial schema
- Basic frontend structure

---

**Phase 2: Core Features (Weeks 3-4)**

**Goal:** Material browsing and file management

**Tasks:**

- [ ] Set up AWS S3 bucket
- [ ] Implement file upload API (admin)
- [ ] Create institution/course/subject hierarchy
- [ ] Build material listing API
- [ ] Create browse page UI
- [ ] Implement material detail page
- [ ] Add download functionality
- [ ] Create filter system
- [ ] Implement basic search
- [ ] Add pagination

**Deliverables:**

- Admin can upload materials
- Students can browse and download
- Working hierarchy navigation
- Basic search functionality

---

**Phase 3: AI Integration (Weeks 5-6)**

**Goal:** Document summarization

**Tasks:**

- [ ] Set up Claude API account
- [ ] Implement text extraction (PDF/PPT)
- [ ] Create AI service module
- [ ] Build summary generation endpoint
- [ ] Create background job queue (optional: use Bull)
- [ ] Store summaries in database
- [ ] Display summaries on frontend
- [ ] Add summary generation UI (admin)
- [ ] Handle errors and retries
- [ ] Add processing status indicators

**Deliverables:**

- Automated summary generation
- Summary display on material pages
- Admin controls for AI features

---

**Phase 4: Enhancement & Polish (Weeks 7-8)**

**Goal:** Improve UX and prepare for launch

**Tasks:**

- [ ] Improve UI/UX design
- [ ] Add loading states
- [ ] Implement error handling
- [ ] Add search filters and sorting
- [ ] Create user dashboard
- [ ] Add analytics (view counts, downloads)
- [ ] Implement tags system
- [ ] Add material recommendations
- [ ] Mobile responsiveness
- [ ] Performance optimization
- [ ] Testing (unit + integration)
- [ ] Write documentation

**Deliverables:**

- Polished, production-ready application
- Good user experience

- Documentation for deployment

---

**Phase 5: Deployment (Week 9)**

**Goal:** Launch MVP

**Tasks:**

- [ ] Set up production database
- [ ] Configure S3 buckets for production
- [ ] Deploy backend to Railway/Render
- [ ] Deploy frontend to Vercel
- [ ] Set up domain and SSL
- [ ] Configure environment variables
- [ ] Set up monitoring (Sentry)
- [ ] Create backup strategy
- [ ] Load testing
- [ ] Security audit

**Deliverables:**

- Live, accessible platform
- Monitoring in place
- Deployment documentation

---

## 8. TECHNICAL IMPLEMENTATION EXAMPLES

### 8.1 Authentication Implementation

**Backend - authController.ts**

typescript

```
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import { Request, Response } from 'express';
import { pool } from '../config/database';

export const register = async (req: Request, res: Response) => {
  try {
    const { email, password, fullName, institutionId } = req.body;

    // Check if user exists
    const existingUser = await pool.query(
      'SELECT * FROM users WHERE email = $1',
      [email]
    );

    if (existingUser.rows.length > 0) {
      return res.status(400).json({
        success: false,
        message: 'Email already registered'
      });
    }

    // Hash password
    const passwordHash = await bcrypt.hash(password, 10);

    // Create user
    const result = await pool.query(
      `INSERT INTO users (email, password_hash, full_name)
       VALUES ($1, $2, $3) RETURNING id, email, full_name`,
      [email, passwordHash, fullName]
    );

    const user = result.rows[0];

    // Create enrollment if institutionId provided
    if (institutionId) {
      await pool.query(
        `INSERT INTO enrollments (user_id, institution_id)
         VALUES ($1, $2)`,
        [user.id, institutionId]
      );
    }

    res.status(201).json({
      success: true,
      message: 'Registration successful',
```

```
      userId: user.id
    });
  } catch (error) {
    console.error('Registration error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

export const login = async (req: Request, res: Response) => {
  try {
    const { email, password } = req.body;

    // Find user
    const result = await pool.query(
      'SELECT * FROM users WHERE email = $1',
      [email]
    );

    if (result.rows.length === 0) {
      return res.status(401).json({
        success: false,
        message: 'Invalid credentials'
      });
    }

    const user = result.rows[0];

    // Verify password
    const isValid = await bcrypt.compare(password, user.password_hash);

    if (!isValid) {
      return res.status(401).json({
        success: false,
        message: 'Invalid credentials'
      });
    }

    // Generate JWT
    const token = jwt.sign(
      { userId: user.id, role: user.role },
      process.env.JWT_SECRET!,
      { expiresIn: '7d' }
    );
```

```
    // Update last login
    await pool.query(
      'UPDATE users SET last_login = CURRENT_TIMESTAMP WHERE id = $1',
      [user.id]
    );

    res.json({
      success: true,
      token,
      user: {
        id: user.id,
        email: user.email,
        fullName: user.full_name,
        role: user.role
      }
    });
  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};
```

### Middleware - auth.ts

typescript

```ts
import jwt from 'jsonwebtoken';
import { Request, Response, NextFunction } from 'express';

interface JwtPayload {
  userId: string;
  role: string;
}

declare global {
  namespace Express {
    interface Request {
      user?: JwtPayload;
    }
  }
}

export const authMiddleware = (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const token = req.headers.authorization?.split(' ')[1];

    if (!token) {
      return res.status(401).json({
        success: false,
        message: 'No token provided'
      });
    }

    const decoded = jwt.verify(
      token,
      process.env.JWT_SECRET!
    ) as JwtPayload;

    req.user = decoded;
    next();
  } catch (error) {
    res.status(401).json({
      success: false,
      message: 'Invalid token'
    });
  }
};
```

```
export const adminMiddleware = (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  if (req.user?.role !== 'admin' && req.user?.role !== 'super_admin') {
    return res.status(403).json({
      success: false,
      message: 'Admin access required'
    });
  }
  next();
};
```

### Frontend - AuthContext.tsx

typescript

```tsx
import React, { createContext, useState, useContext, useEffect } from 'react';
import { authService } from '../services/authService';

interface User {
  id: string;
  email: string;
  fullName: string;
  role: string;
}

interface AuthContextType {
  user: User | null;
  login: (email: string, password: string) => Promise<void>;
  logout: () => void;
  isAuthenticated: boolean;
  isLoading: boolean;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    // Check if token exists and validate
    const token = localStorage.getItem('token');
    if (token) {
      authService.getCurrentUser()
        .then(setUser)
        .catch(() => localStorage.removeItem('token'))
        .finally(() => setIsLoading(false));
    } else {
      setIsLoading(false);
    }
  }, []);

  const login = async (email: string, password: string) => {
    const { token, user } = await authService.login(email, password);
    localStorage.setItem('token', token);
    setUser(user);
  };

  const logout = () => {
    localStorage.removeItem('token');
    setUser(null);
```

```
  };

  return (
    <AuthContext.Provider value={{
      user,
      login,
      logout,
      isAuthenticated: !!user,
      isLoading
    }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) throw new Error('useAuth must be used within AuthProvider');
  return context;
};
```

### 8.2 File Upload Implementation

### Backend - materialController.ts

typescript

```
import { Request, Response } from 'express';
import { pool } from '../config/database';
import { fileService } from '../services/fileService';
import { aiService } from '../services/aiService';

export const uploadMaterial = async (req: Request, res: Response) => {
  try {
    const { title, description, subjectId, type, tags } = req.body;
    const file = req.file;

    if (!file) {
      return res.status(400).json({
        success: false,
        message: 'No file uploaded'
      });
    }

    // Upload to S3
    const fileUrl = await fileService.uploadToS3(
      file,
      `materials/${subjectId}/${Date.now()}-${file.originalname}`
    );

    // Create material record
    const result = await pool.query(
      `INSERT INTO materials
       (subject_id, title, description, type, file_url, file_size, file_name, uploaded_by, tags)
       VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)
       RETURNING *`,
      [
        subjectId,
        title,
        description,
        type,
        fileUrl,
        file.size,
        file.originalname,
        req.user!.userId,
        tags ? tags.split(',') : []
      ]
    );

    const material = result.rows[0];

    // Queue AI summary generation (async)
    aiService.generateSummary(material.id, fileUrl, type)
```

```
      .catch(err => console.error('Summary generation failed:', err));

    res.status(201).json({
      success: true,
      message: 'Material uploaded successfully',
      materialId: material.id,
      processingStatus: 'queued'
    });
  } catch (error) {
    console.error('Upload error:', error);
    res.status(500).json({
      success: false,
      message: 'Upload failed'
    });
  }
};
```

### Backend - fileService.ts

typescript

```typescript
import { S3Client, PutObjectCommand, DeleteObjectCommand } from '@aws-sdk/client-s3';
import { getSignedUrl } from '@aws-sdk/s3-request-presigner';

const s3Client = new S3Client({
  region: process.env.AWS_REGION!,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!
  }
});

export const fileService = {
  async uploadToS3(file: Express.Multer.File, path: string): Promise<string> {
    const command = new PutObjectCommand({
      Bucket: process.env.S3_BUCKET_NAME!,
      Key: path,
      Body: file.buffer,
      ContentType: file.mimetype
    });

    await s3Client.send(command);

    return `https://${process.env.S3_BUCKET_NAME}.s3.${process.env.AWS_REGION}.amazonaws.com/${path}`;
  },

  async deleteFromS3(fileUrl: string): Promise<void> {
    const key = fileUrl.split('.com/')[1];

    const command = new DeleteObjectCommand({
      Bucket: process.env.S3_BUCKET_NAME!,
      Key: key
    });

    await s3Client.send(command);
  }
};
```

### 8.3 AI Summary Generation

### Backend - aiService.ts

typescript

```
import Anthropic from '@anthropic-ai/sdk';
import axios from 'axios';
import pdf from 'pdf-parse';
import { pool } from '../config/database';

const anthropic = new Anthropic({
  apiKey: process.env.CLAUDE_API_KEY!
});

export const aiService = {
  async generateSummary(
    materialId: string,
    fileUrl: string,
    fileType: string
  ): Promise<void> {
    try {
      // 1. Download file
      const response = await axios.get(fileUrl, {
        responseType: 'arraybuffer'
      });

      // 2. Extract text
      let text = '';
      if (fileType === 'pdf') {
        const data = await pdf(response.data);
        text = data.text;
      } else {
        // Handle other types (PPT, etc.)
        text = response.data.toString();
      }

      // Limit text to ~100k tokens (approx 400k chars)
      if (text.length > 400000) {
        text = text.substring(0, 400000);
      }

      // 3. Call Claude API
      const prompt = `You are an educational content summarizer.
Analyze the following study material and provide:

1. A concise summary (150-200 words)
2. 5-7 key points or main concepts
3. List of main topics covered

Material Content:
${text}
```

```
Respond in valid JSON format:
{
  "summary": "...",
  "keyPoints": ["...", "...", "..."],
  "topics": ["...", "..."]
}`;

    const message = await anthropic.messages.create({
      model: 'claude-3-sonnet-20240229',
      max_tokens: 2048,
      messages: [{
        role: 'user',
        content: prompt
      }]
    });

    // 4. Parse response
    const responseText = message.content[0].type === 'text'
      ? message.content[0].text
      : '';

    const summaryData = JSON.parse(responseText);

    // 5. Save to database
    await pool.query(
      `INSERT INTO ai_summaries
       (material_id, summary_text, key_points, model_used)
       VALUES ($1, $2, $3, $4)`,
      [
        materialId,
        summaryData.summary,
        summaryData.keyPoints,
        'claude-3-sonnet-20240229'
      ]
    );

    // 6. Update material status
    await pool.query(
      'UPDATE materials SET is_processed = true WHERE id = $1',
      [materialId]
    );

    console.log(`Summary generated for material ${materialId}`);
  } catch (error) {
    console.error('AI summary generation error:', error);
    throw error;
```

```
  }
 }
};
```

---

## 9. DEPLOYMENT GUIDE

### Environment Variables

### Backend (.env)

```
# Database
DATABASE_URL=postgresql://user:password@host:5432/dbname

# JWT
JWT_SECRET=your-super-secret-key-change-this

# AWS S3
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=your-access-key
AWS_SECRET_ACCESS_KEY=your-secret-key
S3_BUCKET_NAME=study-platform-materials

# Claude AI
CLAUDE_API_KEY=your-claude-api-key

# Server
PORT=5000
NODE_ENV=production

# CORS
FRONTEND_URL=https://yourdomain.com
```

### Frontend (.env)

```
REACT_APP_API_URL=https://api.yourdomain.com/v1
```

### Deployment Steps

### 1. Database (Railway/Supabase)

bash

```
# Create PostgreSQL instance
# Run migrations
psql $DATABASE_URL < schema.sql
```

### 2. Backend (Railway/Render)

```
bash

# Connect GitHub repo
# Set environment variables
# Deploy
```

### 3. Frontend (Vercel)

```
bash

# Install Vercel CLI
npm i -g vercel

# Deploy
vercel --prod
```

### 4. S3 Setup

```
bash

# Create S3 bucket
# Set bucket policy for public read
# Enable CORS
```

---

## 10. TESTING STRATEGY

### Unit Tests

- Authentication functions
- File upload logic
- AI service functions
- Database queries

### Integration Tests

- API endpoints
- Authentication flow

- File upload and download
- Search functionality

**Tools**

- Jest for backend
- React Testing Library for frontend
- Supertest for API testing

---

## 11. SECURITY CONSIDERATIONS

1. **Authentication & Authorization**
    - JWT with expiration
    - Password hashing with bcrypt
    - Role-based access control
    - Secure cookie storage
2. **File Upload**
    - File type validation
    - File size limits (e.g., 50MB)
    - Virus scanning (optional)
    - Sanitize file names
3. **API Security**
    - Rate limiting (express-rate-limit)
    - CORS configuration
    - Input validation
    - SQL injection prevention (parameterized queries)
    - XSS protection
4. **Data Protection**
    - HTTPS only
    - Environment variables for secrets
    - Database encryption at rest
    - Secure S3 bucket policies

---

## 12. COST ESTIMATION (Monthly)

**MVP Scale (100 users, 1000 materials)**

- **Database (Railway/Supabase):** $10-20
- **Backend Hosting (Railway):** $5-10
- **Frontend Hosting (Vercel):** Free
- **S3 Storage (10GB):** $0.23
- **S3 Bandwidth (100GB):** $9
- **Claude API (1000 summaries):** ~$30-50
- **Domain:** $12/year

**Total: ~$55-90/month**

**At Scale (10,000 users, 100,000 materials)**

- **Database:** $50-100
- **Backend:** $50-100
- **S3 Storage (1TB):** $23
- **S3 Bandwidth (10TB):** $900
- **Claude API:** $500-1000
- **CDN (CloudFront):** $50

**Total: ~$1,600-2,200/month**

---

## 13. FUTURE ENHANCEMENTS (Post-MVP)

**Phase 3 Features**

- Interactive chatbot for Q&A
- Video materials support
- Real-time collaboration
- Discussion forums
- Bookmarking and favorites
- Progress tracking
- Personalized recommendations

**Advanced Features**

- Mobile apps (React Native)

- Offline mode
- OCR for scanned documents
- Multiple language support
- Integration with LMS platforms
- Analytics dashboard
- Gamification (badges, points)

---

## 14. LEARNING RESOURCES

**For You (CSE 2nd Year)**

**React & TypeScript:**

- Official React docs: https://react.dev
- TypeScript handbook: https://www.typescriptlang.org/docs/

**Node.js & Express:**

- Express.js guide: https://expressjs.com/
- Node.js best practices: https://github.com/goldbergyoni/nodebestpractices

**PostgreSQL:**

- PostgreSQL tutorial: https://www.postgresqltutorial.com/

**AWS S3:**

- AWS S3 docs: https://docs.aws.amazon.com/s3/

**Claude API:**

- Anthropic docs: https://docs.anthropic.com/

---

## 15. SUCCESS METRICS

**Launch Metrics (First Month)**

- 50+ registered users
- 100+ materials uploaded
- 500+ downloads
- 90%+ AI summary accuracy (subjective review)

**Growth Metrics (3 Months)**

- 500+ users
- 1000+ materials
- 10,000+ downloads
- 3+ institutions onboarded

---

## NEXT STEPS

1. **Week 1 Action Items:**
    - Set up Git repository
    - Initialize React and Node.js projects
    - Create PostgreSQL database
    - Implement basic authentication
2. **Questions to Answer:**
    - Which institutions will you start with?
    - Do you have AWS/Claude API credits?
    - Will you build alone or with a team?
    - Do you have a target launch date?
3. **Tools to Set Up:**
    - VS Code (IDE)
    - PostgreSQL (local)
    - Postman (API testing)
    - Git & GitHub
    - Node.js & npm

---

**This plan is comprehensive but flexible. Start with Phase 1 and adjust as you learn. Focus on building a working MVP first, then iterate based on user feedback.**

Good luck with your project! 🚀