🎓 StudyMate AI - Complete Project Specifications

📋 TABLE OF CONTENTS

---

1. PROJECT OVERVIEW

1.1 Project Name

StudyMate AI - Intelligent Study Companion Platform

1.2 Team

- **Backend Developer (You):** NestJS API, Database, AI Integration
- **Frontend Developer (Partner):** Angular UI, Components, User Experience

1.3 Project Goal

A web platform where university students can:

- ✅ Organize subjects and courses
- ✅ Upload multiple PDFs per course with tags
- ✅ Use AI to summarize PDFs
- ✅ Take AI-generated quizzes with automatic scoring
- ✅ Chat with AI (global context-aware chat)
- ✅ Manage exam/DS/assignment calendar
- ✅ Track study statistics and streaks

1.4 Tech Stack

```
Backend:   NestJS + TypeScript + TypeORM
Database:  PostgreSQL + pgAdmin
Frontend:  Angular 20 + TailwindCSS
AI:        Hugging Face API
DevOps:    Docker + Docker Compose
Version:   Git + GitHub
```

1.5 Example User Journey

Day 1: Setup

- Student creates account
- Creates subject "Mathematics"
- Adds course "Chapter 1: Algebra"
- Uploads 3 PDFs: notes.pdf, exercises.pdf, textbook.pdf
- Tags PDFs: "Important", "Lecture", "Practice"

Day 2: Study

- Opens notes.pdf
- Clicks "Summarize" → AI gives 3-paragraph summary
- Opens global AI chat
- Asks: "Explain quadratic equations" (AI remembers previous context)
- Attaches textbook.pdf to chat → "Summarize page 15-20 of this PDF"

Day 3: Quiz

- Clicks "Generate Quiz" for Chapter 1
- AI creates 10 questions (multiple choice, true/false, short answer)
- Student answers all questions
- AI checks answers → Score: 8/10 (80%)
- Cannot retake same quiz

Day 4: Calendar

- Adds exam: "Math Midterm - November 15"
- Dashboard shows: "Math Midterm in 10 days"

- Next day dashboard shows: "Math Midterm in 9 days"

Week 1: Statistics

- Dashboard shows:
  - 3 subjects, 8 courses, 24 PDFs
  - 5 quizzes taken, average: 75%
  - Study streak: 7 days 🔥
  - Most studied: Mathematics

---

2. FUNCTIONAL REQUIREMENTS

2.1 User Management

FR-001: User Registration

- Email and password required
- Password min 8 characters
- Email validation
- System creates user account

FR-002: User Authentication

- Login with email/password
- JWT token generated (expires in 7 days)
- Token stored securely in frontend
- Auto-logout on token expiry

FR-003: User Profile

- View profile (name, email, join date)
- Update name
- Change password
- View total stats

2.2 Subject Management

FR-004: Create Subject

- Student creates subject (e.g., "Mathematics")
- Required: Name
- Optional: Color (hex code), Semester, Professor
- Color auto-assigned if not chosen

FR-005: View Subjects

- Dashboard shows all subjects as cards
- Each card displays:
  - Subject name and color
  - Number of courses
  - Number of PDFs
  - Next upcoming deadline (if any)
- Can search/filter subjects

FR-006: Edit Subject

- Modify name, color, semester, professor
- Changes save immediately

FR-007: Delete Subject

- Confirmation modal: "Delete subject and ALL courses/PDFs inside?"
- If confirmed: Cascade delete (removes courses, PDFs, quizzes)

2.3 Course Management

FR-008: Create Course

- Create course inside a subject
- Required: Name (e.g., "Chapter 1: Algebra")
- Optional: Description, Order number
- Automatically linked to parent subject

FR-009: View Courses

- Click subject → See all courses inside
- Courses display as list or cards

- Each course shows:
  - Course name
  - Number of PDFs
  - Last studied date
  - Quiz count
- Sorted by order number or creation date

FR-010: Edit Course

- Modify name, description, order
- Reorder courses (drag-and-drop optional)

FR-011: Delete Course

- Confirmation: "Delete course and all PDFs?"
- Cascade delete (removes PDFs, quizzes)

2.4 PDF Management

FR-012: Upload PDFs

- Upload multiple PDFs to a course
- Max file size: 10MB per PDF
- Accepted format: PDF only
- Can upload multiple files at once

FR-013: PDF Metadata

- Each PDF has:
  - Filename (can be renamed)
  - Description/Notes (optional text)
  - Tags (e.g., "Important", "Lecture", "Exam Material")
  - Upload date
  - File size
  - Last accessed date

FR-014: Organize PDFs

- Add multiple tags per PDF

- Filter PDFs by tag
- Search PDFs by name or description
- Sort by: name, date, size

FR-015: View PDF

- View PDF in browser (embedded viewer)
- Download original PDF
- See PDF details (size, upload date, tags)

FR-016: Edit PDF

- Rename PDF
- Edit description
- Add/remove tags
- Cannot edit PDF content (only metadata)

FR-017: Delete PDF

- Confirmation modal
- Also deletes related AI summaries and quizzes

2.5 AI Summarization

FR-018: Generate Summary

- Student clicks "Summarize" on a PDF
- System extracts text from PDF
- Sends text to Hugging Face API
- AI returns 3-4 paragraph summary
- Summary saved to database
- Display summary in modal or dedicated page

FR-019: View Summaries

- Each PDF can have multiple summaries (if regenerated)
- Show creation date for each summary
- Can delete old summaries

FR-020: Summary History

- Dashboard shows recent summaries
- Can view all summaries ever generated
- Filter by subject/course

2.6 AI Quiz System

FR-021: Generate Quiz

- Student clicks "Generate Quiz" on a PDF or course
- AI analyzes PDF content
- Creates 10 questions:
  - Multiple Choice (4 options)
  - True/False
  - Short Answer
- Mix of question types
- Quiz saved with unique ID

FR-022: Take Quiz

- Student sees questions one by one (or all at once)
- Select answers for MC and T/F
- Type answers for short answer
- Submit quiz when complete

FR-023: AI Grading

- System sends student answers to AI
- AI checks answers:
  - MC/T/F: Exact match
  - Short answer: AI evaluates correctness
- Returns score (e.g., 8/10 = 80%)
- Shows correct answers
- Highlights wrong answers

FR-024: Quiz Results

- Display score prominently
- Show breakdown:
  - Question 1: ✅ Correct
  - Question 2: ❌ Wrong (Correct answer: X)
- Save result to database
- Cannot retake same quiz

FR-025: Quiz History

- View all quizzes taken
- See scores and dates
- Filter by subject/course
- View average score
- Can review questions and answers

2.7 Global AI Chat

FR-026: Chat Interface

- Global chat accessible from anywhere
- Not limited to specific course/PDF
- Chat interface shows:
  - User messages
  - AI responses
  - Timestamps
  - Attached PDFs (if any)

FR-027: Context-Aware Chat

- AI remembers conversation context
- Can reference previous messages
- Example:
  - User: "What is algebra?"
  - AI: "Algebra is..."
  - User: "Give me an example" (AI knows "example" refers to algebra)

FR-028: Attach PDF to Chat

- User can attach a PDF to message
- AI reads PDF content
- User can ask: "Summarize this PDF"
- User can ask: "What's on page 5?"

FR-029: Chat Actions

- Ask questions about any subject
- Request explanations
- Get study tips
- Ask for examples
- Compare concepts

FR-030: Chat History

- All conversations saved
- Can scroll through history
- Search chat history
- Can start new conversation (separate threads optional)

2.8 Calendar & Deadlines

FR-031: Add Event

- Create calendar event:
  - **Type:** Exam, DS (Devoir Surveillé), Assignment
  - **Title:** (e.g., "Math Midterm")
  - **Date:** Date only (no time)
  - **Subject:** Link to subject (not specific course)
  - **Description:** (optional)

FR-032: View Calendar

- Calendar view (month grid)
- Shows all events

- Color-coded by subject
- Click event to see details

FR-033: Upcoming Deadlines

- Dashboard shows next 7 days' events
- Sorted by date
- Shows countdown: "Math Exam in 3 days"

FR-034: Notifications

- Dashboard highlights:
  - Events tomorrow: "⚠️ Math DS tomorrow!"
  - Events today: "🔴 Chemistry Exam TODAY!"
- Visual indicators (colors, icons)

FR-035: Edit/Delete Events

- Modify event details
- Delete event with confirmation

2.9 Dashboard & Statistics

FR-036: Dashboard Overview Shows:

- **Quick Stats Card:**
  - Total subjects: 5
  - Total courses: 23
  - Total PDFs: 87
- **Quiz Performance Card:**
  - Total quizzes taken: 15
  - Average score: 78%
  - Last quiz: Math Ch.1 - 9/10
- **Study Streak Card:**
  - Current streak: 12 days 🔥
  - Longest streak: 20 days
  - Streak explained: "Days with activity (upload, quiz, chat, summary)"

- **Most Studied Subject:**
  - Mathematics (45 PDFs, 8 quizzes)
- **Recent Activity:**
  - "You summarized Physics Ch.2 - 2 hours ago"
  - "You took quiz on Math Ch.1 - Score: 9/10 - Yesterday"
  - "You chatted with AI about Chemistry - 3 days ago"
- **Upcoming Deadlines:**
  - Math DS - November 5 (in 3 days)
  - Physics Assignment - November 8 (in 6 days)

FR-037: Study Streak Calculation

- Day counts if student does ANY of:
  - Uploads PDF
  - Takes quiz
  - Generates summary
  - Chats with AI
  - Views course/PDF
- Streak breaks if no activity for 24 hours
- Shows current streak and longest streak

FR-038: Detailed Statistics Page

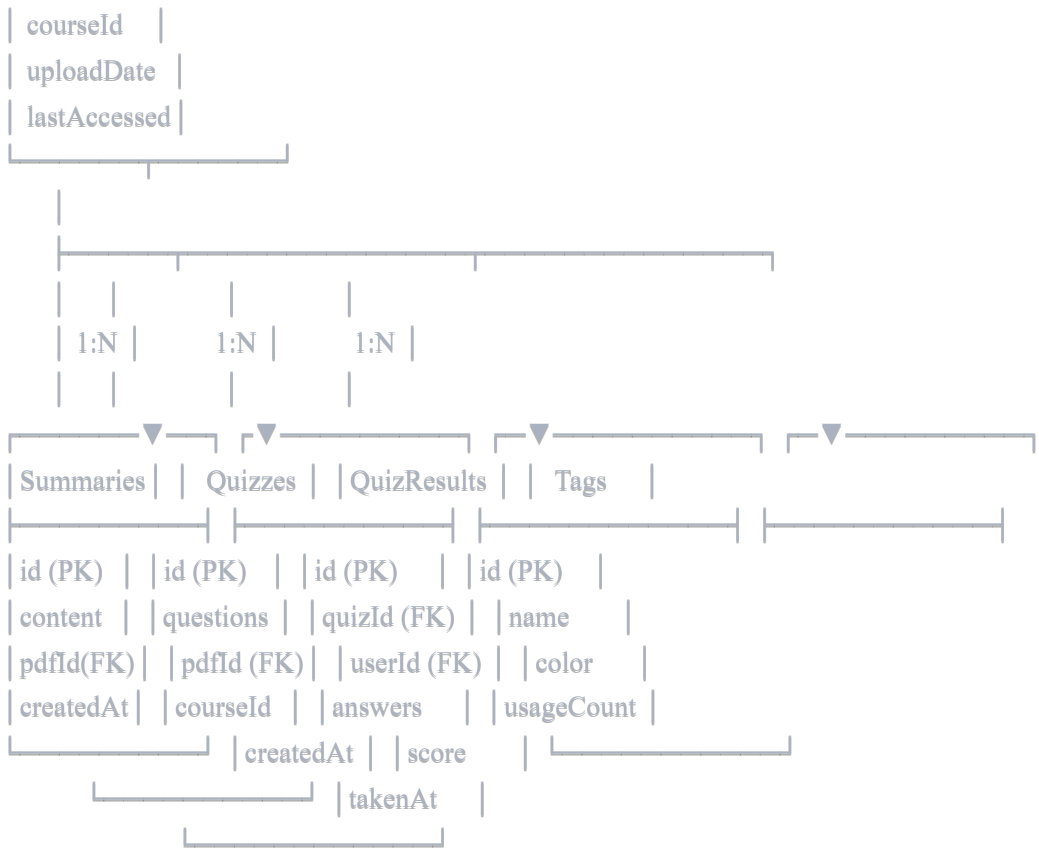
- Charts and graphs:
  - Quiz scores over time (line chart)
  - PDFs per subject (bar chart)
  - Study activity heatmap (calendar grid)
  - Most active days/times
- Filterable by date range

---

3. DATABASE DESIGN

3.1 Entity Relationship Diagram (ERD)


Users
id (PK)
email
passwordHash
fullName
createdAt
lastActive
1:N
Subjects
id (PK)
name
color
semester
professor
userId (FK)
Events
id (PK)
title
type
date
subjectId
userId (FK)
ChatHistory
id (PK)
message
response
pdfFileId
userId (FK)
StudyStreak
id (PK)
userId (FK)
date
activityType
createdAt
1:N
Courses
id (PK)
name
description
orderNumber
subjectId
createdAt
lastStudied
1:N
PdfFiles
id (PK)
fileName
filePath
fileSize
description
tags
(JSON array)



3.2 Detailed Database Tables

Users Table

```typescript
{
  id: UUID PRIMARY KEY,
  email: VARCHAR(255) UNIQUE NOT NULL,
  passwordHash: VARCHAR(255) NOT NULL,
  fullName: VARCHAR(100) NOT NULL,
  createdAt: TIMESTAMP DEFAULT NOW(),
  lastActive: TIMESTAMP,
  longestStreak: INTEGER DEFAULT 0,
  currentStreak: INTEGER DEFAULT 0
}
```

Subjects Table

```typescript
```

```
{
  id: UUID PRIMARY KEY,
  name: VARCHAR(100) NOT NULL,
  color: VARCHAR(7) NOT NULL, // Hex color
  semester: VARCHAR(50),
  professor: VARCHAR(100),
  userId: UUID FOREIGN KEY → Users(id) ON DELETE CASCADE,
  createdAt: TIMESTAMP DEFAULT NOW()
}
```

Courses Table

typescript

```
{
  id: UUID PRIMARY KEY,
  name: VARCHAR(200) NOT NULL,
  description: TEXT,
  orderNumber: INTEGER DEFAULT 0,
  subjectId: UUID FOREIGN KEY → Subjects(id) ON DELETE CASCADE,
  createdAt: TIMESTAMP DEFAULT NOW(),
  lastStudied: TIMESTAMP
}
```

PdfFiles Table

typescript

```
{
  id: UUID PRIMARY KEY,
  fileName: VARCHAR(200) NOT NULL,
  filePath: VARCHAR(500) NOT NULL, // Server storage path
  fileSize: BIGINT NOT NULL, // Bytes
  description: TEXT, // User notes about PDF
  tags: JSONB, // Array of tag names: ["Important", "Lecture"]
  courseId: UUID FOREIGN KEY → Courses(id) ON DELETE CASCADE,
  uploadDate: TIMESTAMP DEFAULT NOW(),
  lastAccessed: TIMESTAMP
}
```

Tags Table (Reusable tags)

typescript

```
{
  id: UUID PRIMARY KEY,
  name: VARCHAR(50) UNIQUE NOT NULL,
  color: VARCHAR(7),
  userId: UUID FOREIGN KEY → Users(id) ON DELETE CASCADE,
  usageCount: INTEGER DEFAULT 0 // How many PDFs use this tag
}
```

Summaries Table

typescript

```
{
  id: UUID PRIMARY KEY,
  content: TEXT NOT NULL, // AI-generated summary
  pdfFileId: UUID FOREIGN KEY → PdfFiles(id) ON DELETE CASCADE,
  createdAt: TIMESTAMP DEFAULT NOW()
}
```

Quizzes Table

typescript

```
{
 id: UUID PRIMARY KEY,
 questions: JSONB NOT NULL,
 /* Example structure:
 [
  {
   type: "multiple_choice",
   question: "What is 2+2?",
   options: ["2", "3", "4", "5"],
   correctAnswer: "4"
  },
  {
   type: "true_false",
   question: "The sky is blue",
   correctAnswer: true
  },
  {
   type: "short_answer",
   question: "Explain photosynthesis",
   correctAnswer: "Process where plants convert light to energy..."
  }
 ]
 */
 pdfFileId: UUID FOREIGN KEY → PdfFiles(id) ON DELETE CASCADE,
 courseId: UUID FOREIGN KEY → Courses(id) ON DELETE CASCADE,
 createdAt: TIMESTAMP DEFAULT NOW()
}
```

QuizResults Table

typescript

```
{
 id: UUID PRIMARY KEY,
 quizId: UUID FOREIGN KEY → Quizzes(id) ON DELETE CASCADE,
 userId: UUID FOREIGN KEY → Users(id) ON DELETE CASCADE,
 answers: JSONB NOT NULL, // Student's answers
 score: DECIMAL(5,2) NOT NULL, // Percentage: 80.00
 correctCount: INTEGER NOT NULL, // 8/10
 totalQuestions: INTEGER NOT NULL, // 10
 takenAt: TIMESTAMP DEFAULT NOW()
}
```

Events Table (Calendar)

```typescript
{
  id: UUID PRIMARY KEY,
  title: VARCHAR(200) NOT NULL,
  type: VARCHAR(20) NOT NULL, // "exam", "ds", "assignment"
  date: DATE NOT NULL,
  description: TEXT,
  subjectId: UUID FOREIGN KEY → Subjects(id) ON DELETE SET NULL,
  userId: UUID FOREIGN KEY → Users(id) ON DELETE CASCADE,
  createdAt: TIMESTAMP DEFAULT NOW()
}
```

ChatHistory Table

```typescript
{
  id: UUID PRIMARY KEY,
  message: TEXT NOT NULL, // User's message
  response: TEXT NOT NULL, // AI's response
  pdfFileId: UUID FOREIGN KEY → PdfFiles(id) ON DELETE SET NULL, // If PDF attached
  userId: UUID FOREIGN KEY → Users(id) ON DELETE CASCADE,
  createdAt: TIMESTAMP DEFAULT NOW()
}
```

StudyStreak Table

```typescript
{
  id: UUID PRIMARY KEY,
  userId: UUID FOREIGN KEY → Users(id) ON DELETE CASCADE,
  date: DATE NOT NULL,
  activityType: VARCHAR(50) NOT NULL, // "upload", "quiz", "summary", "chat"
  createdAt: TIMESTAMP DEFAULT NOW(),
  UNIQUE(userId, date) // One record per user per day
}
```

---

4. API SPECIFICATIONS

4.1 Base URL

- Development: `http://localhost:3000/api`

- Production: https://studymate.app/api

4.2 Authentication

All endpoints (except register/login) require:

```
Headers: {
  Authorization: "Bearer <jwt-token>"
}
```

4.3 API Endpoints

Authentication

POST /api/auth/register

typescript

```typescript
// Request
{
  email: string;
  password: string;
  fullName: string;
}

// Response (201)
{
  user: {
    id: string;
    email: string;
    fullName: string;
  },
  token: string;
}
```

POST /api/auth/login

typescript

```typescript
// Request
{
  email: string;
  password: string;
}

// Response (200)
{
  user: {
    id: string;
    email: string;
    fullName: string;
  },
  token: string;
}
```

GET /api/auth/profile

typescript

```typescript
// Response (200)
{
  id: string;
  email: string;
  fullName: string;
  currentStreak: number;
  longestStreak: number;
  createdAt: string;
}
```

Subjects

GET /api/subjects

typescript

```typescript
// Response (200)
[
  {
    id: string;
    name: string;
    color: string;
    semester: string;
    professor: string;
    coursesCount: number;
    pdfsCount: number;
    nextDeadline: {
      title: string;
      date: string;
      daysUntil: number;
    } | null;
  }
]
```

POST /api/subjects

typescript

```typescript
// Request
{
  name: string;
  color?: string; // Optional, auto-assigned if not provided
  semester?: string;
  professor?: string;
}

// Response (201)
{
  id: string;
  name: string;
  color: string;
  semester: string;
  professor: string;
  createdAt: string;
}
```

PUT /api/subjects/:id DELETE /api/subjects/:id

Courses

GET /api/courses?subjectId=:id

```typescript
// Response (200)
[
  {
    id: string;
    name: string;
    description: string;
    orderNumber: number;
    subjectId: string;
    pdfsCount: number;
    quizzesCount: number;
    lastStudied: string | null;
    createdAt: string;
  }
]
```

POST /api/courses

```typescript
// Request
{
  name: string;
  description?: string;
  orderNumber?: number;
  subjectId: string;
}

// Response (201)
{
  id: string;
  name: string;
  description: string;
  orderNumber: number;
  subjectId: string;
}
```

PUT /api/courses/:id DELETE /api/courses/:id

PDF Files

GET /api/pdf-files?courseId=:id&tags=tag1,tag2

```typescript
```

```typescript
// Response (200)
[
  {
    id: string;
    fileName: string;
    fileSize: number;
    description: string;
    tags: string[];
    courseId: string;
    uploadDate: string;
    lastAccessed: string;
    summariesCount: number;
    hasQuiz: boolean;
  }
]
```

POST /api/pdf-files

typescript

```typescript
// Request (multipart/form-data)
{
  file: File; // PDF file
  courseId: string;
  description?: string;
  tags?: string[]; // ["Important", "Lecture"]
}

// Response (201)
{
  id: string;
  fileName: string;
  fileSize: number;
  description: string;
  tags: string[];
  courseId: string;
  uploadDate: string;
}
```

PUT /api/pdf-files/:id

typescript

```typescript
// Request
{
  fileName?: string;
  description?: string;
  tags?: string[];
}
```

GET /api/pdf-files/:id/download

```
Response: Binary PDF file
```

DELETE /api/pdf-files/:id

Tags

GET /api/tags

typescript

```typescript
// Response (200)
[
  {
    id: string;
    name: string;
    color: string;
    usageCount: number;
  }
]
```

POST /api/tags

typescript

```typescript
// Request
{
  name: string;
  color?: string;
}
```

AI - Summaries

POST /api/ai/summarize

```typescript
// Request
{
  pdfFileId: string;
}

// Response (200)
{
  summary: string; // AI-generated summary
  savedAt: string;
}
```

GET /api/summaries?pdfFileId=:id

```typescript
// Response (200)
[
  {
    id: string;
    content: string;
    pdfFileId: string;
    createdAt: string;
  }
]
```

AI - Quizzes

POST /api/ai/generate-quiz

```typescript
```

```
// Request
{
  pdfFileId: string;
  // OR
  courseId: string; // Generate from all PDFs in course
}

// Response (201)
{
  quizId: string;
  questions: [
    {
      id: string;
      type: "multiple_choice" | "true_false" | "short_answer";
      question: string;
      options?: string[]; // For multiple choice
      // correctAnswer NOT sent to frontend
    }
  ];
}
```

POST /api/quizzes/:id/submit

typescript

```
// Request
{
  answers: [
    {
      questionId: string;
      answer: string | boolean;
    }
  ];
}

// Response (200)
{
  score: number; // Percentage: 80
  correctCount: number; // 8
  totalQuestions: number; // 10
  results: [
    {
      questionId: string;
      question: string;
      yourAnswer: string;
      correctAnswer: string;
      isCorrect: boolean;
    }
  ];
}
```

GET /api/quiz-results

typescript

```
// Response (200)
[
  {
    id: string;
    quizId: string;
    score: number;
    correctCount: number;
    totalQuestions: number;
    takenAt: string;
    courseName: string;
    subjectName: string;
  }
]
```

GET /api/quiz-results/stats

```typescript
// Response (200)
{
  totalQuizzes: number;
  averageScore: number;
  highestScore: number;
  lowestScore: number;
  scoresBySubject: {
    [subjectName: string]: number; // Average score
  };
}
```

AI - Chat

POST /api/ai/chat

```typescript
// Request
{
  message: string;
  pdfFileId?: string; // Optional: attach PDF
  conversationHistory?: {
    message: string;
    response: string;
  }[]; // For context
}

// Response (200)
{
  response: string; // AI response
  savedAt: string;
}
```

GET /api/chat-history

```typescript
```

```typescript
// Response (200)
[
  {
    id: string;
    message: string;
    response: string;
    pdfFileName: string | null;
    createdAt: string;
  }
]
```

Calendar Events

GET /api/events?from=2024-11-01&to=2024-11-30

typescript

```typescript
// Response (200)
[
  {
    id: string;
    title: string;
    type: "exam" | "ds" | "assignment";
    date: string; // "2024-11-15"
    description: string;
    subjectId: string;
    subjectName: string;
    subjectColor: string;
    daysUntil: number; // Calculated
  }
]
```

POST /api/events

typescript

```typescript
// Request
{
  title: string;
  type: "exam" | "ds" | "assignment";
  date: string; // "2024-11-15"
  description?: string;
  subjectId: string;
}
```

PUT /api/events/:id DELETE /api/events/:id

GET /api/events/upcoming

```typescript
// Returns next 7 days' events
// Response same as GET /api/events
```

Dashboard & Statistics

GET /api/dashboard

```typescript
```

```typescript
// Response (200)
{
  stats: {
    totalSubjects: number;
    totalCourses: number;
    totalPdfs: number;
    totalQuizzes: number;
    averageScore: number;
    currentStreak: number;
    longestStreak: number;
  },
  mostStudiedSubject: {
    id: string;
    name: string;
    pdfsCount: number;
    quizzesCount: number;
  },
  recentActivity: [
    {
      type: "summary" | "quiz" | "chat" | "upload";
      description: string;
      timestamp: string;
    }
  ],
  upcomingDeadlines: [
    {
      title: string;
      date: string;
      type: string;
      daysUntil: number;
      subjectName: string;
    }
  ];
}
```

GET /api/statistics/detailed

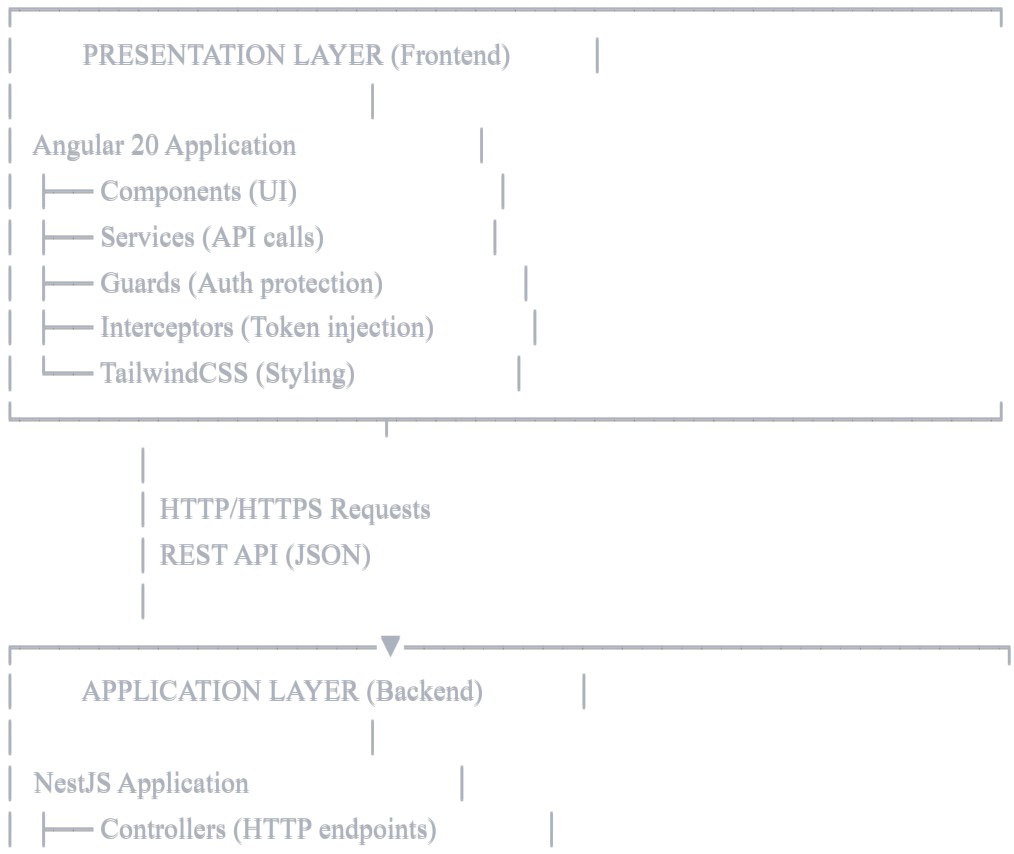
typescript

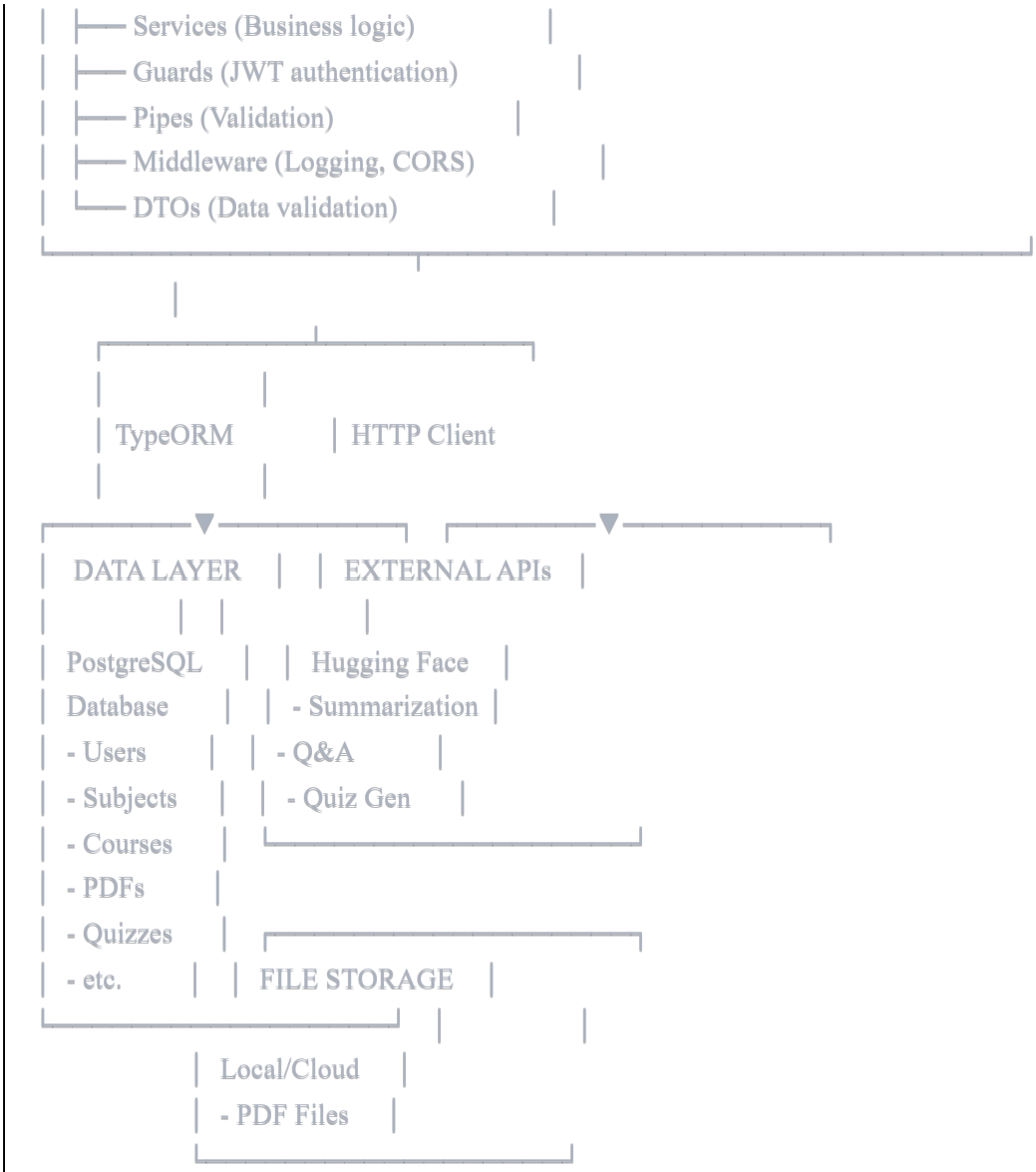
```
// Response (200)
{
 quizScoresOverTime: [
  {
   date: string;
   score: number;
  }
 ],
 pdfsPerSubject: {
  [subjectName: string]: number;
 },
 studyActivityHeatmap: [
  {
   date: string;
   activityCount: number;
  }
 ],
 mostActiveDay: string;
 mostActiveTime: string;
}
```

---

5. SYSTEM ARCHITECTURE

5.1 Architecture Diagram





5.2 Technology Details

Frontend Stack

- **Angular 20**: Component-based UI framework
- **TypeScript**: Type-safe JavaScript
- **TailwindCSS**: Utility-first CSS framework
- **RxJS**: Reactive programming for async operations
- **Angular Router**: Navigation
- **HttpClient**: API communication
- **ng2-pdf-viewer**: PDF viewing in browser
- **Chart.js**: Data visualization for statistics

Backend Stack

- **NestJS**: Progressive Node.js framework

- **TypeScript**: Type-safe development
- **TypeORM**: ORM for database operations
- **Passport.js**: Authentication strategies
- **JWT**: JSON Web Tokens for auth
- **bcrypt**: Password hashing
- **class-validator**: DTO validation
- **multer**: File upload handling
- **pdf-parse**: Extract text from PDFs

Database

- **PostgreSQL 16**: Relational database
- **pgAdmin 4**: Database management GUI

DevOps

- **Docker**: Containerization
- **Docker Compose**: Multi-container orchestration
- **Git**: Version control
- **GitHub**: Code repository

---

6. PROJECT STRUCTURE

6.1 Repository Structure

```
studymate/
├── backend/                # NestJS Backend
│   ├── src/
│   │   ├── modules/        # Feature modules
│   │   │   ├── auth/
│   │   │   ├── subjects/
│   │   │   ├── courses/
│   │   │   ├── pdf-files/
│   │   │   ├── ai/
│   │   │   ├── quizzes/
│   │   │   ├── events/
│   │   │   ├── dashboard/
│   │   │   └── chat/
│   │   ├── common/         # Shared code
```

```
│   │   │   ├── guards/
│   │   │   ├── decorators/
│   │   │   ├── filters/
│   │   │   └── pipes/
│   │   ├── database/          # DB config
│   │   ├── config/            # App config
│   │   └── main.ts            # Entry point
│   ├── uploads/               # PDF storage
│   ├── package.json
│   ├── tsconfig.json
│   ├── Dockerfile
│   └── .env
│
├── frontend/                  # Angular Frontend
│   ├── src/
│   │   ├── app/
│   │   │   ├── core/          # Singleton services
│   │   │   │   ├── services/
│   │   │   │   ├── guards/
│   │   │   │   └── interceptors/
│   │   │   ├── shared/        # Shared components
│   │   │   │   ├── components/
│   │   │   │   ├── pipes/
│   │   │   │   └── directives/
│   │   │   ├── features/      # Feature modules
│   │   │   │   ├── auth/
│   │   │   │   ├── dashboard/
│   │   │   │   ├── subjects/
│   │   │   │   ├── courses/
│   │   │   │   ├── pdf-viewer/
│   │   │   │   ├── quiz/
│   │   │   │   ├── chat/
│   │   │   │   ├── calendar/
│   │   │   │   └── statistics/
│   │   │   ├── app.component.ts
│   │   │   └── app.routes.ts
│   │   ├── assets/
│   │   ├── styles/
│   │   └── environments/
│   ├── package.json
│   ├── angular.json
│   ├── tailwind.config.js
│   ├── Dockerfile
│   └── .env
│
├── docker-compose.yml         # Orchestration
```

```
├── .gitignore
└── README.md
```

6.2 Backend Module Structure (Example: Subjects)

```
backend/src/modules/subjects/
├── dto/
│   ├── create-subject.dto.ts
│   ├── update-subject.dto.ts
│   └── subject-response.dto.ts
├── entities/
│   └── subject.entity.ts
├── subjects.controller.ts
├── subjects.service.ts
└── subjects.module.ts
```

6.3 Frontend Feature Structure (Example: Subjects)

```
frontend/src/app/features/subjects/
├── components/
│   ├── subject-list/
│   │   ├── subject-list.component.ts
│   │   ├── subject-list.component.html
│   │   └── subject-list.component.css
│   ├── subject-card/
│   ├── subject-form/
│   └── subject-detail/
├── services/
│   └── subjects.service.ts
├── models/
│   └── subject.model.ts
└── subjects.routes.ts
```

---

7. STEP-BY-STEP DEVELOPMENT GUIDE

🎯 WEEK-BY-WEEK BREAKDOWN

WEEK 1: Project Setup & Environment

Day 1-2: Install Everything

You (Backend):

1. Install Node.js (v18+) from nodejs.org
2. Install PostgreSQL 16 + pgAdmin
3. Install Visual Studio Code
4. Install Git
5. Install Docker Desktop
6. Install Postman (for API testing)

Commands to verify:

```bash
node --version  # Should show v18+
npm --version
psql --version
docker --version
git --version
```

Her (Frontend):

1. Install Node.js (v18+)
2. Install Visual Studio Code
3. Install Git
4. Install Angular CLI: `npm install -g @angular/cli`

Commands to verify:

```bash
node --version
npm --version
ng version  # Should show Angular CLI
git --version
```

Day 3: Create GitHub Repository

Together:

1. One person creates GitHub repo: `studymate-ai`
2. Add other person as collaborator
3. Clone repo locally:

```bash
git clone https://github.com/username/studymate-ai.git
cd studymate-ai
```

Day 4: Backend Setup

You (Backend):

```bash
# Create NestJS project
npm i -g @nestjs/cli
nest new backend
cd backend

# Install dependencies
npm install @nestjs/typeorm typeorm pg
npm install @nestjs/passport passport passport-jwt
npm install @nestjs/jwt bcrypt
npm install class-validator class-transformer
npm install @nestjs/config
npm install multer @types/multer
npm install pdf-parse axios

# Create folder structure
mkdir src/modules
mkdir src/common
mkdir uploads
```

Create `.env` file:

```env
DATABASE_HOST=localhost
DATABASE_PORT=5432
DATABASE_USER=postgres
DATABASE_PASSWORD=your_password
DATABASE_NAME=studymate
JWT_SECRET=your-super-secret-key-change-this
JWT_EXPIRATION=7d
HUGGINGFACE_API_KEY=your-hf-api-key
```

Day 5: Frontend Setup

Her (Frontend):

```bash
# Create Angular project
cd ..
ng new frontend --routing --style=css
cd frontend

# Install dependencies
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init

# Install additional packages
npm install @angular/material
npm install chart.js ng2-charts
npm install ngx-extended-pdf-viewer
```

Configure TailwindCSS in `tailwind.config.js`:

```javascript
module.exports = {
  content: [
    "./src/**/*.{html,ts}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Day 6-7: Docker Setup

Together:

Create `docker-compose.yml` in root:

```yaml
```

```yaml
version: '3.8'

services:
  database:
    image: postgres:16
    container_name: studymate-db
    environment:
      POSTGRES_USER: studymate_user
      POSTGRES_PASSWORD: studymate_pass
      POSTGRES_DB: studymate_db
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - studymate-network

  pgadmin:
    image: dpage/pgadmin4
    container_name: studymate-pgadmin
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@studymate.com
      PGADMIN_DEFAULT_PASSWORD: admin123
    ports:
      - "5050:80"
    networks:
      - studymate-network
    depends_on:
      - database

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: studymate-backend
    environment:
      DATABASE_HOST: database
      DATABASE_PORT: 5432
      DATABASE_USER: studymate_user
      DATABASE_PASSWORD: studymate_pass
      DATABASE_NAME: studymate_db
    ports:
      - "3000:3000"
    volumes:
      - ./backend:/app
      - /app/node_modules
```

```
    networks:
      - studymate-network
    depends_on:
      - database

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: studymate-frontend
    ports:
      - "4200:4200"
    volumes:
      - ./frontend:/app
      - /app/node_modules
    networks:
      - studymate-network
    depends_on:
      - backend

volumes:
  postgres_data:

networks:
  studymate-network:
    driver: bridge
```

Create `backend/Dockerfile`:

dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "run", "start:dev"]
```

Create `frontend/Dockerfile`:

dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 4200
CMD ["npm", "start"]
```

Test Docker:

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

Commit everything:

```bash
git add .
git commit -m "Initial project setup with Docker"
git push origin main
```

WEEK 2: Database & Authentication

Day 1-2: Database Setup

You (Backend):

1. Create database entities in `src/modules/`

User Entity (`auth/entities/user.entity.ts`):

```typescript
```

```
import { Entity, PrimaryGeneratedColumn, Column, CreateDateColumn, OneToMany } from 'typeorm';
import { Subject } from '../../subjects/entities/subject.entity';

@Entity('users')
export class User {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column({ unique: true })
  email: string;

  @Column()
  passwordHash: string;

  @Column()
  fullName: string;

  @CreateDateColumn()
  createdAt: Date;

  @Column({ type: 'timestamp', nullable: true })
  lastActive: Date;

  @Column({ default: 0 })
  currentStreak: number;

  @Column({ default: 0 })
  longestStreak: number;

  @OneToMany(() => Subject, subject => subject.user)
  subjects: Subject[];
}
```

2. Set up database connection in `app.module.ts`:

```
typescript
```

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ConfigModule } from '@nestjs/config';

@Module({
  imports: [
    ConfigModule.forRoot(),
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DATABASE_HOST,
      port: parseInt(process.env.DATABASE_PORT),
      username: process.env.DATABASE_USER,
      password: process.env.DATABASE_PASSWORD,
      database: process.env.DATABASE_NAME,
      entities: [__dirname + '/**/*.entity{.ts,.js}'],
      synchronize: true, // ONLY for development!
    }),
  ],
})
export class AppModule {}
```

3. Run backend to create tables:

```
bash

npm run start:dev
```

4. Verify in pgAdmin:
   - Open http://localhost:5050
   - Login with admin@studymate.com / admin123
   - Connect to database
   - Check if `users` table exists

Day 3-4: Backend Authentication

You (Backend):

Create auth module:

```
bash

nest g module auth
nest g controller auth
nest g service auth
```

Implement JWT strategy, register, login endpoints.

Key files:

- `auth/dto/register.dto.ts`
- `auth/dto/login.dto.ts`
- `auth/auth.service.ts` (bcrypt hashing, JWT signing)
- `auth/auth.controller.ts` (POST /register, POST /login)
- `auth/strategies/jwt.strategy.ts`
- `auth/guards/jwt-auth.guard.ts`

Day 5-7: Frontend Authentication

Her (Frontend):

1. Create auth module:

```bash
ng generate module features/auth --routing
ng generate component features/auth/login
ng generate component features/auth/register
ng generate service features/auth/services/auth
```

2. Implement:
   - Login form with validation
   - Register form with validation
   - Auth service (API calls)
   - Auth guard (protect routes)
   - Token interceptor (add JWT to requests)
3. Style with TailwindCSS

Test together:

- Register new user
- Login with credentials
- Verify JWT token is stored
- Test protected route access

WEEK 3: Subjects & Courses

Day 1-2: Backend - Subjects

You (Backend):

```bash
nest g module subjects
nest g controller subjects
nest g service subjects
```

Create:

- Subject entity with relationships
- DTOs for create/update
- CRUD operations in service
- Controller endpoints

Day 3-4: Frontend - Subjects

Her (Frontend):

```bash
ng generate module features/subjects --routing
ng generate component features/subjects/subject-list
ng generate component features/subjects/subject-card
ng generate component features/subjects/subject-form
ng generate service features/subjects/services/subjects
```

Create:

- Subject list page (grid of cards)
- Subject form modal (create/edit)
- Delete confirmation
- Color picker for subjects

Day 5-6: Backend - Courses

You (Backend):

```bash
```

```
nest g module courses
nest g controller courses
nest g service courses
```

Create:

- Course entity (relationship to Subject)
- CRUD operations
- Filter by subject

Day 7: Frontend - Courses

Her (Frontend):

```bash
ng generate module features/courses --routing
ng generate component features/courses/course-list
ng generate component features/courses/course-form
ng generate service features/courses/services/courses
```

WEEK 4: PDF Upload & Management

Day 1-3: Backend - PDF Upload

You (Backend):

```bash
nest g module pdf-files
nest g controller pdf-files
nest g service pdf-files
```

Implement:

- Multer configuration for file upload
- PDF entity with tags (JSONB)
- Upload endpoint (multipart/form-data)
- Download endpoint
- PDF metadata management
- Tag system

Key code:

```typescript
@Post('upload')
@UseInterceptors(FileInterceptor('file'))
async uploadPdf(
  @UploadedFile() file: Express.Multer.File,
  @Body() dto: UploadPdfDto,
  @GetUser() user: User,
) {
  // Validate file type
  // Save file to disk/cloud
  // Create database record
  // Extract text for AI (optional)
}
```

Day 4-7: Frontend - PDF Management

Her (Frontend):

```bash
ng generate module features/pdf-viewer --routing
ng generate component features/pdf-viewer/pdf-list
ng generate component features/pdf-viewer/pdf-upload
ng generate component features/pdf-viewer/pdf-viewer
ng generate component features/pdf-viewer/pdf-card
ng generate service features/pdf-viewer/services/pdf
```

Implement:

- Drag-and-drop upload
- Multiple file upload
- PDF list with thumbnails
- Tag management UI
- PDF viewer (ngx-extended-pdf-viewer)
- Download button
- Edit metadata (name, description, tags)

WEEK 5: AI Integration - Summarization

Day 1-3: Backend - AI Service

You (Backend):

```bash
nest g module ai
nest g controller ai
nest g service ai
```

Create Hugging Face integration:

```typescript
// ai/ai.service.ts
import axios from 'axios';

@Injectable()
export class AiService {
  private readonly hfApiKey = process.env.HUGGINGFACE_API_KEY;
  private readonly summarizationModel = 'facebook/bart-large-cnn';

  async summarize(text: string): Promise<string> {
    const response = await axios.post(
      `https://api-inference.huggingface.co/models/${this.summarizationModel}`,
      { inputs: text },
      {
        headers: {
          Authorization: `Bearer ${this.hfApiKey}`,
        },
      },
    );
    return response.data[0].summary_text;
  }
}
```

Create:

- PDF text extraction (pdf-parse)
- POST /ai/summarize endpoint
- Summary entity and storage
- GET /summaries/:pdfId endpoint

Day 4-7: Frontend - Summarization UI

Her (Frontend):

```bash
ng generate component features/pdf-viewer/pdf-summary
ng generate service features/ai/services/ai
```

Implement:

- "Summarize" button on PDF viewer
- Loading spinner during AI processing
- Display summary in modal or card
- Save/view multiple summaries
- Recent summaries on dashboard

WEEK 6: AI Integration - Quiz System

Day 1-4: Backend - Quiz Generation

You (Backend):

Create quiz generation:

```typescript
// ai/ai.service.ts
async generateQuiz(text: string): Promise<QuizQuestion[]> {
  // Call Hugging Face for question generation
  // Parse response into structured questions
  // Mix question types: MC, T/F, Short Answer
  // Return 10 questions
}
```

Create:

- Quiz entity (with questions as JSONB)
- QuizResult entity
- POST /ai/generate-quiz
- POST /quizzes/:id/submit (AI grades answers)
- GET /quiz-results (history)

Implement AI grading:

- Multiple choice: exact match
- True/False: exact match
- Short answer: Use AI to check correctness

Day 5-7: Frontend - Quiz UI

Her (Frontend):

```bash
ng generate module features/quiz --routing
ng generate component features/quiz/quiz-take
ng generate component features/quiz/quiz-result
ng generate component features/quiz/quiz-history
ng generate service features/quiz/services/quiz
```

Implement:

- "Generate Quiz" button
- Quiz taking interface:
  - Multiple choice with radio buttons
  - True/False with buttons
  - Short answer with text input
- Submit quiz
- Show results with:
  - Score (big and prominent)
  - Correct/wrong answers highlighted
  - Correct answers shown
- Quiz history page
- Cannot retake message

WEEK 7: Global AI Chat

Day 1-3: Backend - Chat System

You (Backend):

```bash
nest g module chat
nest g controller chat
nest g service chat
```

Implement:

- ChatHistory entity
- POST /ai/chat endpoint
- Context handling (send last N messages)
- PDF attachment handling
- GET /chat-history

```typescript
async chat(
  message: string,
  pdfFileId?: string,
  conversationHistory?: ChatMessage[],
): Promise<string> {
  let context = '';

  // Add conversation history for context
  if (conversationHistory) {
    context += conversationHistory.map(m =>
      `User: ${m.message}\nAI: ${m.response}`
    ).join('\n');
  }

  // Add PDF content if attached
  if (pdfFileId) {
    const pdfText = await this.extractPdfText(pdfFileId);
    context += `\nPDF Content: ${pdfText}`;
  }

  // Send to Hugging Face
  const response = await this.callHuggingFace(context + '\n' + message);
  return response;
}
```

Day 4-7: Frontend - Chat Interface

Her (Frontend):

```bash
ng generate module features/chat --routing
ng generate component features/chat/chat-window
ng generate component features/chat/chat-message
ng generate component features/chat/chat-input
ng generate service features/chat/services/chat
```

Implement:

- Chat interface (like WhatsApp/Messenger)
- Message bubbles (user vs AI)
- Attach PDF button
- Send message
- Loading indicator
- Auto-scroll to latest message
- Context preservation (send last 10 messages)
- Search chat history

WEEK 8: Calendar & Events

Day 1-3: Backend - Events

You (Backend):

```bash
nest g module events
nest g controller events
nest g service events
```

Create:

- Event entity
- CRUD operations
- GET /events/upcoming (next 7 days)
- Filter by date range
- Calculate daysUntil

Day 4-7: Frontend - Calendar

Her (Frontend):

```bash
ng generate module features/calendar --routing
ng generate component features/calendar/calendar-view
ng generate component features/calendar/event-form
ng generate component features/calendar/event-list
ng generate service features/calendar/services/events
```

Implement:

- Calendar grid (can use library or custom)
- Add event form
- Event types (Exam, DS, Assignment)
- Color-code by subject
- Upcoming deadlines widget
- Notifications on dashboard:
  - "Exam tomorrow!"
  - "DS in 3 days"

WEEK 9: Dashboard & Statistics

Day 1-3: Backend - Dashboard API

You (Backend):

```bash
nest g module dashboard
nest g controller dashboard
nest g service dashboard
```

Create endpoints:

- GET /dashboard (all stats)
- GET /statistics/detailed
- Calculate study streak

- Recent activity tracking
- Most studied subject

Day 4-7: Frontend - Dashboard & Stats

Her (Frontend):

```bash
ng generate component features/dashboard/dashboard-overview
ng generate component features/dashboard/stats-card
ng generate component features/dashboard/activity-feed
ng generate component features/dashboard/streak-display
ng generate module features/statistics --routing
ng generate component features/statistics/statistics-page
ng generate component features/statistics/charts
```

Implement:

- Dashboard overview page
- Stats cards:
  - Total subjects, courses, PDFs
  - Quiz performance
  - Study streak with fire emoji 🔥
- Recent activity feed
- Upcoming deadlines widget
- Most studied subject
- Charts (Chart.js):
  - Quiz scores over time (line chart)
  - PDFs per subject (bar chart)
  - Study heatmap

WEEK 10: Testing, Bug Fixes & Polish

Day 1-2: Testing

You (Backend):

- Test all API endpoints with Postman

- Write unit tests for critical services
- Test file upload edge cases
- Test AI integration
- Check database constraints

Her (Frontend):

- Test all user flows
- Cross-browser testing
- Mobile responsive testing
- Test error scenarios
- Check loading states

Day 3-4: Bug Fixes

Together:

- Create bug list
- Prioritize (critical → nice-to-have)
- Fix bugs
- Test fixes

Day 5-6: UI/UX Polish

Her (Frontend):

- Improve animations
- Add transitions
- Improve error messages
- Add empty states ("No PDFs yet")
- Add success toasts
- Improve loading indicators

Day 7: Final Review

Together:

- Full app walkthrough
- Check all features work

- Test complete user journey
- Verify Docker deployment
- Prepare for presentation

WEEK 11-12: Documentation & PFE Report

See Section 9 below for detailed report structure.

---

8. TASK DIVISION

👨‍💻 YOUR RESPONSIBILITIES (Backend Developer)

Core Backend Tasks:

1. ✅ NestJS project setup
2. ✅ Database design and entities (TypeORM)
3. ✅ All API endpoints and controllers
4. ✅ Authentication (JWT, bcrypt)
5. ✅ File upload system (Multer)
6. ✅ PDF text extraction
7. ✅ Hugging Face AI integration
8. ✅ Quiz generation algorithm
9. ✅ AI grading logic
10. ✅ Database migrations
11. ✅ API testing (Postman)
12. ✅ Docker configuration for backend
13. ✅ Environment configuration

Shared Tasks:

1. ✅ Docker Compose setup
2. ✅ Git repository management
3. ✅ API contract design (endpoints structure)
4. ✅ Database schema design

5. ✅ Integration testing
6. ✅ Deployment

Documentation Tasks:

1. ✅ API documentation (Swagger/Postman)
2. ✅ Database ERD diagrams
3. ✅ Backend architecture documentation
4. ✅ Installation guide (backend)

👩‍💻 HER RESPONSIBILITIES (Frontend Developer)

Core Frontend Tasks:

1. ✅ Angular project setup
2. ✅ All UI components
3. ✅ TailwindCSS styling
4. ✅ Responsive design
5. ✅ Forms and validation
6. ✅ API integration (services)
7. ✅ Routing and navigation
8. ✅ Auth guard implementation
9. ✅ PDF viewer integration
10. ✅ Charts and data visualization
11. ✅ Calendar component
12. ✅ File upload UI (drag-and-drop)
13. ✅ Loading states and animations
14. ✅ Error handling UI

Shared Tasks:

1. ✅ User flow design
2. ✅ Feature testing
3. ✅ Bug fixing

4. ✅ UI/UX improvements

Documentation Tasks:

1. ✅ User manual (with screenshots)
2. ✅ UI/UX design documentation
3. ✅ Frontend architecture documentation
4. ✅ Component documentation

🤝 COLLABORATION WORKFLOW

Daily (10 minutes):

- Morning standup (video call or message)
- Share: What did you finish? What are you working on? Any blockers?

Code Reviews:

- All Pull Requests must be reviewed
- Review within 24 hours
- No direct pushes to `main` branch

Git Workflow:

```bash
```

```
# Your workflow
git checkout -b feature/backend-auth
# ... make changes ...
git add .
git commit -m "Add: JWT authentication"
git push origin feature/backend-auth
# Create Pull Request on GitHub
# She reviews and approves
# Merge to main

# Her workflow
git checkout -b feature/frontend-login
# ... make changes ...
git commit -m "Add: Login page UI"
git push origin feature/frontend-login
# Create Pull Request
# You review and approve
# Merge to main
```

Communication:

- **WhatsApp/Discord:** Quick questions
- **GitHub Issues:** Track tasks and bugs
- **GitHub Projects:** Kanban board (To Do, In Progress, Done)
- **Weekly meeting:** 30-60 min review and planning

When Stuck:

1. Try to solve for 30 minutes
2. Google the error
3. Ask your partner
4. Ask ChatGPT/Claude
5. Check Stack Overflow
6. Document the solution (for PFE report!)

---

9. PFE REPORT STRUCTURE

📄 Report Outline (80-120 pages)

Chapter 1: General Introduction (10-15 pages)

1.1. Context

- Digital transformation in education
- Need for intelligent study tools
- AI in education landscape

1.2. Problem Statement

- Students struggle with information overload
- Lack of personalized study tools
- Difficulty organizing study materials

1.3. Proposed Solution

- StudyMate AI platform description
- Key features overview
- Expected benefits

1.4. Project Objectives

- Organize study materials
- AI-powered learning assistance
- Performance tracking
- Deadline management

1.5. Project Methodology

- Agile development approach
- Sprint-based development
- Tools and technologies

1.6. Report Structure

- Chapter summaries

Chapter 2: State of the Art (15-20 pages)

2.1. Existing Solutions Analysis

- Notion (general organization)
- Quizlet (quizzes)

- Evernote (notes)
- Google Classroom
- Canvas LMS
- **Comparison table with StudyMate**

2.2. Technologies Study

- **Frontend Frameworks:**
  - React vs Angular vs Vue
  - Why Angular 20?
- **Backend Frameworks:**
  - Express vs NestJS vs Fastify
  - Why NestJS?
- **Databases:**
  - SQL vs NoSQL
  - PostgreSQL vs MySQL vs MongoDB
  - Why PostgreSQL?
- **AI/ML Services:**
  - OpenAI vs Hugging Face vs Google AI
  - Why Hugging Face?