

Admin Material Upload Web App - Frontend Developer Guide

Overview

This guide provides complete API documentation for building an admin web application to manage educational content, including hierarchical navigation, question upload, resource management, and advanced question fetching with Q-matrix display.

Base URL: `http://localhost:5200/api`

Table of Contents

- 1. [Dashboard with Content Count](#)
 - 2. [Hierarchical Creation Workflow](#)
 - 3. [Question Upload Service](#)
 - 4. [PYQ Upload & Session Builder](#)
 - 5. [Resource Upload Service](#)
 - 6. [Enhanced Question Fetching](#)
 - 7. [Complete API Reference](#)
-

1. Dashboard with Content Count

Page: Dashboard Overview

Purpose: Display content statistics filtered by hierarchy level

API Endpoints

Get All Exams with Question Counts

```
GET /api/hierarchy/exams
```

Response:

```
[
  {
    "id": "exam-uuid-1",
    "name": "JEE Main",
    "description": "Joint Entrance Examination",
    "exam_type": "competitive",
    "created_at": "2025-01-01T00:00:00Z"
  }
]
```

Get Question Count for Exam

```
GET /api/hierarchy/exam/{exam_id}/question-count
```

Response:

```
{
  "exam_id": "exam-uuid-1",
  "total_question_count": 1500
}
```

Get Question Count for Subject

```
GET /api/hierarchy/subject/{subject_id}/question-count
```

Response:

```
{
  "subject_id": "subject-uuid-1",
  "total_question_count": 500
}
```

Get Question Count for Chapter

```
GET /api/hierarchy/chapter/{chapter_id}/question-count
```

Response:

```
{
  "chapter_id": "chapter-uuid-1",
  "total_question_count": 100
}
```

Get Question Count for Topic

```
GET /api/hierarchy/topic/{topic_id}/question-count
```

Response:

```
{
  "topic_id": "topic-uuid-1",
  "total_question_count": 25
}
```

Get Statistics for Any Level

```
GET /api/hierarchy/{level}/{item_id}/stats
```

Parameters:

- **level**: exam, subject, chapter, topic, concept

Response:

```
{
  "question_count": 150,
  "attribute_count": 12,
  "difficulty_avg": 0.65,
  "discrimination_avg": 1.2,
  "guessing_avg": 0.25,
  "attributes": [
    {
      "id": "attr-uuid-1",
      "name": "Understanding Newton's Laws",
      "question_count": 45
    }
  ],
  "difficulty_distribution": {
    "very_easy": 10,
    "easy": 30,
    "medium": 60,
    "hard": 40,
    "very_hard": 10
  }
}
```

Frontend Implementation Example

```
// Dashboard Component
interface DashboardStats {
  exams: ExamWithCount[];
```

```
selectedLevel: 'exam' | 'subject' | 'chapter' | 'topic';
stats: HierarchyStats | null;
}

async function fetchDashboardStats(): Promise<DashboardStats> {
  // Get all exams
  const exams = await fetch('/api/hierarchy/exams').then(r => r.json());

  // For each exam, get question count
  const examsWithCounts = await Promise.all(
    exams.map(async (exam) => {
      const count = await fetch(`/api/hierarchy/exam/${exam.id}/question-
count`)
        .then(r => r.json());
      return { ...exam, questionCount: count.total_question_count };
    })
  );

  return { exams: examsWithCounts, selectedLevel: 'exam', stats: null };
}

// Get detailed stats for selected item
async function getItemStats(level: string, itemId: string) {
  const response = await fetch(`/api/hierarchy/${level}/${itemId}/stats`);
  return response.json();
}
```

2. Hierarchical Creation Workflow

Page: Content Creation

Purpose: Create hierarchical structure with TWO different paths

Two Hierarchy Paths

Path 1: Competitive Exam (JEE, NEET, etc.)

Exam (competitive) → Subject → Chapter → Topic → Concept

Path 2: School Exam (CBSE, ICSE, etc.)

Exam (school) → Class → Subject → Chapter → Topic → Concept

2.1 Create Exam

POST /api/exams

Request Body:

```
{
  "name": "JEE Main 2025",
  "description": "Joint Entrance Examination for Engineering",
  "exam_type": "competitive"
}
```

Exam Types:

- **"competitive"** - For competitive exams (JEE, NEET, CAT, etc.)
 - Goes directly to subjects
 - No class level
- **"school"** - For school exams (CBSE, ICSE, State Board, etc.)
 - Requires class level after exam
 - Then subjects under class

Response:

```
[{
  "id": "exam-uuid-1",
  "name": "JEE Main 2025",
  "description": "Joint Entrance Examination for Engineering",
  "exam_type": "competitive",
  "created_at": "2025-01-01T00:00:00Z"
}]
```

Example - School Exam:

```
{
  "name": "CBSE Board",
  "description": "Central Board of Secondary Education",
  "exam_type": "school"
}
```

2.2 Create Class (Required ONLY for School Exams)

⚠ **IMPORTANT:** This step is ONLY for **exam_type: "school"**

POST /api/classes

Request Body:

```
{
  "exam_id": "exam-uuid-1",
  "name": "Class 10",
  "description": "Class 10 CBSE",
  "class_number": 10,
  "section": "A"
}
```

Response:

```
[{
  "id": "class-uuid-1",
  "exam_id": "exam-uuid-1",
  "name": "Class 10",
  "description": "Class 10 CBSE",
  "class_number": 10,
  "section": "A",
  "created_at": "2025-01-01T00:00:00Z"
}]
```

Get Classes for a School Exam:

```
GET /api/hierarchy/classes?exam_id={exam_id}
```

Response:

```
[
  {
    "id": "class-uuid-1",
    "exam_id": "exam-uuid-1",
    "name": "Class 10",
    "class_number": 10,
    "section": "A"
  },
  {
    "id": "class-uuid-2",
    "exam_id": "exam-uuid-1",
    "name": "Class 11",
    "class_number": 11,
    "section": "Science"
  }
]
```

2.3 Create Subject

 **For Competitive Exam Path:** Subject is directly under Exam (no class)

```
POST /api/subjects
```

Request Body:

```
{
  "exam_id": "exam-uuid-1",
  "name": "Physics",
  "description": "Physics for JEE Main"
}
```

 **For School Exam Path:** Subject is under Class (must provide class_id, not exam_id)

```
POST /api/subjects
```

Request Body:

```
{
  "class_id": "class-uuid-1",
  "name": "Physics",
  "description": "Physics for Class 10"
}
```

Response:

```
[{
  "id": "subject-uuid-1",
  "exam_id": "exam-uuid-1",
  "class_id": "class-uuid-1",
  "name": "Physics",
  "description": "Physics for Class 10",
  "created_at": "2025-01-01T00:00:00Z"
}]
```

Get Subjects:

For Competitive Exam:

```
GET /api/hierarchy/subjects?exam_id={exam_id}
```

For School Exam:

```
GET /api/hierarchy/subjects?class_id={class_id}
```

2.4 Create Chapter

```
POST /api/chapters
```

Request Body:

```
{
  "subject_id": "subject-uuid-1",
  "name": "Mechanics",
  "description": "Laws of Motion and Dynamics"
}
```

Response:

```
[{
  "id": "chapter-uuid-1",
  "subject_id": "subject-uuid-1",
  "name": "Mechanics",
  "description": "Laws of Motion and Dynamics",
  "created_at": "2025-01-01T00:00:00Z"
}]
```

Get Chapters:

```
GET /api/hierarchy/chapters?subject_id={subject_id}
```

2.5 Create Topic

```
POST /api/topics
```


Request Body:

```
{
  "chapter_id": "chapter-uuid-1",
  "name": "Newton's Laws of Motion",
  "description": "Three fundamental laws of motion"
}
```

Response:

```
[{
  "id": "topic-uuid-1",
  "chapter_id": "chapter-uuid-1",
  "name": "Newton's Laws of Motion",
  "description": "Three fundamental laws of motion",
  "created_at": "2025-01-01T00:00:00Z"
}]
```

Get Topics:

```
GET /api/hierarchy/topics?chapter_id={chapter_id}
```

2.6 Create Concept (Optional)

```
POST /api/concepts
```

Request Body:

```
{
  "topic_id": "topic-uuid-1",
  "name": "First Law of Motion",
  "description": "Law of Inertia"
}
```

Response:

```
[{
  "id": "concept-uuid-1",
  "topic_id": "topic-uuid-1",
  "name": "First Law of Motion",
  "description": "Law of Inertia",
}
```

```
"created_at": "2025-01-01T00:00:00Z"
}]
```

Get Concepts:

```
GET /api/hierarchy/concepts?topic_id={topic_id}
```

2.7 Get Hierarchy Tree

```
GET /api/hierarchy/tree
```

Response:

```
[
  {
    "id": "exam-uuid-1",
    "name": "JEE Main 2025",
    "type": "exam",
    "exam_type": "competitive",
    "children": [
      {
        "id": "subject-uuid-1",
        "name": "Physics",
        "type": "subject",
        "children": [
          {
            "id": "chapter-uuid-1",
            "name": "Mechanics",
            "type": "chapter",
            "children": [
              {
                "id": "topic-uuid-1",
                "name": "Newton's Laws",
                "type": "topic",
                "children": [
                  {
                    "id": "concept-uuid-1",
                    "name": "First Law",
                    "type": "concept"
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
  }
]
```

```
}
]
```

Frontend Implementation Example

```
// Hierarchical Creation Component with TWO PATHS
interface HierarchyState {
  selectedExam: Exam | null;
  selectedClass: Class | null; // Only for school exams
  selectedSubject: Subject | null;
  selectedChapter: Chapter | null;
  selectedTopic: Topic | null;
}

interface Exam {
  id: string;
  name: string;
  exam_type: 'competitive' | 'school';
}

// Cascading dropdown implementation with conditional class level
const HierarchySelector: React.FC = () => {
  const [state, setState] = useState<HierarchyState>({
    selectedExam: null,
    selectedClass: null,
    selectedSubject: null,
    selectedChapter: null,
    selectedTopic: null
  });

  const [exams, setExams] = useState<Exam[]>([]);
  const [classes, setClasses] = useState<Class[]>([]);
  const [subjects, setSubjects] = useState<Subject[]>([]);
  const [chapters, setChapters] = useState<Chapter[]>([]);
  const [topics, setTopics] = useState<Topic[]>([]);

  // Check if selected exam is school type
  const isSchoolExam = state.selectedExam?.exam_type === 'school';

  // Load classes when SCHOOL exam is selected
  useEffect(() => {
    if (state.selectedExam && isSchoolExam) {
      fetch(`/api/hierarchy/classes?exam_id=${state.selectedExam.id}`)
        .then(r => r.json())
        .then(data => setClasses(data));
    }

    // Reset dependent selections
    setState(prev => ({
      ...prev,
      selectedClass: null,
      selectedSubject: null,
    }));
  }, [state.selectedExam]);
}
```

```
        selectedChapter: null,
        selectedTopic: null
    }));
}
}, [state.selectedExam]);

// Load subjects based on exam type
useEffect(() => {
    if (state.selectedExam) {
        if (isSchoolExam && state.selectedClass) {
            // School path: Load subjects under class
            fetch(`/api/hierarchy/subjects?
class_id=${state.selectedClass.id}`)
                .then(r => r.json())
                .then(data => setSubjects(data));
        } else if (!isSchoolExam) {
            // Competitive path: Load subjects under exam
            fetch(`/api/hierarchy/subjects?exam_id=${state.selectedExam.id}`)
                .then(r => r.json())
                .then(data => setSubjects(data));
        }

        // Reset dependent selections
        setState(prev => ({
            ...prev,
            selectedSubject: null,
            selectedChapter: null,
            selectedTopic: null
        }));
    }
}, [state.selectedExam, state.selectedClass, isSchoolExam]);

// Load chapters when subject is selected
useEffect(() => {
    if (state.selectedSubject) {
        fetch(`/api/hierarchy/chapters?
subject_id=${state.selectedSubject.id}`)
            .then(r => r.json())
            .then(data => setChapters(data));

        setState(prev => ({
            ...prev,
            selectedChapter: null,
            selectedTopic: null
        }));
    }
}, [state.selectedSubject]);

// Load topics when chapter is selected
useEffect(() => {
    if (state.selectedChapter) {
        fetch(`/api/hierarchy/topics?
chapter_id=${state.selectedChapter.id}`)
            .then(r => r.json())
```

```

        .then(data => setTopics(data));

        setState(prev => ({ ...prev, selectedTopic: null }));
    }
}, [state.selectedChapter]);

return (
    <div className="hierarchy-selector">
        { /* Exam Selector */ }
        <select onChange={e => {
            const exam = exams.find(ex => ex.id === e.target.value);
            setState(prev => ({ ...prev, selectedExam: exam || null }));
        }}>
            <option value="">Select Exam...</option>
            {exams.map(exam => (
                <option key={exam.id} value={exam.id}>
                    {exam.name} ({exam.exam_type})
                </option>
            ))}
        </select>

        { /* Class Selector – Only show for school exams */ }
        {isSchoolExam && (
            <select
                onChange={e => {
                    const cls = classes.find(c => c.id === e.target.value);
                    setState(prev => ({ ...prev, selectedClass: cls || null }));
                }}
                disabled={!state.selectedExam}
            >
                <option value="">Select Class...</option>
                {classes.map(cls => (
                    <option key={cls.id} value={cls.id}>
                        {cls.name}
                    </option>
                ))}
            </select>
        )}

        { /* Subject Selector */ }
        <select
            onChange={e => {
                const subject = subjects.find(s => s.id === e.target.value);
                setState(prev => ({ ...prev, selectedSubject: subject || null
            }));
            }}
            disabled={
                !state.selectedExam ||
                (isSchoolExam && !state.selectedClass)
            }
        >
            <option value="">Select Subject...</option>
            {subjects.map(subject => (
                <option key={subject.id} value={subject.id}>

```

```

        {subject.name}
      </option>
    )}}
  </select>

  {/* Chapter Selector */}
  <select
    onChange={e => {
      const chapter = chapters.find(c => c.id === e.target.value);
      setState(prev => ({ ...prev, selectedChapter: chapter || null
    }));
    }}
    disabled={!state.selectedSubject}
  >
    <option value="">Select Chapter...</option>
    {chapters.map(chapter => (
      <option key={chapter.id} value={chapter.id}>
        {chapter.name}
      </option>
    ))}
  </select>

  {/* Topic Selector */}
  <select
    onChange={e => {
      const topic = topics.find(t => t.id === e.target.value);
      setState(prev => ({ ...prev, selectedTopic: topic || null }));
    }}
    disabled={!state.selectedChapter}
  >
    <option value="">Select Topic...</option>
    {topics.map(topic => (
      <option key={topic.id} value={topic.id}>
        {topic.name}
      </option>
    ))}
  </select>

  {/* Breadcrumb Display */}
  <div className="breadcrumb">
    {state.selectedExam && (
      <>
        <span>{state.selectedExam.name}</span>
        {isSchoolExam && state.selectedClass && (
          <>
            <span> → </span>
            <span>{state.selectedClass.name}</span>
          </>
        )}
        {state.selectedSubject && (
          <>
            <span> → </span>
            <span>{state.selectedSubject.name}</span>
          </>
        )}
      </>
    )}
  </div>

```

```

    })
    {state.selectedChapter && (
      <>
      <span> → </span>
      <span>{state.selectedChapter.name}</span>
    </>
    )}
    {state.selectedTopic && (
      <>
      <span> → </span>
      <span>{state.selectedTopic.name}</span>
    </>
    )}
  </>
  )}
</div>
</div>
);
};

// Create new subject - handles both paths
async function createSubject(examId: string, examType: 'competitive' |
'school', classId?: string, name: string, description: string) {
  const data = examType === 'school'
    ? { class_id: classId, name, description } // School path
    : { exam_id: examId, name, description }; // Competitive path

  const response = await fetch('/api/subjects', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  });
  return response.json();
}

// Create new class (only for school exams)
async function createClass(examId: string, name: string, classNumber:
number, section?: string) {
  const response = await fetch('/api/classes', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      exam_id: examId,
      name,
      class_number: classNumber,
      section,
      description: `${name} ${section || ''}`
    })
  });
  return response.json();
}

```

3. Question Upload Service

Page: Question Upload

Purpose: Upload questions with images, manage questions (create, edit, delete)

3.1 Upload Single Question with Attributes

POST /api/questions/create-with-attributes

Request Body:

```
{
  "question": {
    "content": "What is Newton's First Law of Motion?",
    "options": {
      "A": "Law of Inertia",
      "B": "F = ma",
      "C": "Action-Reaction",
      "D": "None"
    },
    "correct_answer": "A",
    "exam_id": "exam-uuid-1",
    "subject_id": "subject-uuid-1",
    "chapter_id": "chapter-uuid-1",
    "topic_id": "topic-uuid-1",
    "concept_id": "concept-uuid-1",
    "difficulty": 0.5,
    "discrimination": 1.2,
    "guessing": 0.25,
    "metadata": {
      "marks": 4,
      "time_limit": 120
    }
  },
  "selected_attributes": [
    {
      "attribute_id": "attr-uuid-1",
      "value": true
    },
    {
      "attribute_id": "attr-uuid-2",
      "value": true
    }
  ],
  "create_new_attributes": [
    {
      "name": "Understanding Newton's First Law",
      "description": "Ability to explain law of inertia"
    }
  ]
}
```



```
]
}
```

Response:

```
{
  "question": {
    "id": "question-uuid-1",
    "content": "What is Newton's First Law of Motion?",
    "options": { "A": "Law of Inertia", "B": "F = ma", "C": "Action-
Reaction", "D": "None" },
    "correct_answer": "A",
    "difficulty": 0.5,
    "created_at": "2025-01-01T00:00:00Z"
  },
  "parameters": {
    "difficulty": 0.5,
    "discrimination": 1.2,
    "guessing": 0.25
  },
  "selected_attributes_count": 2,
  "created_attributes": [...],
  "q_matrix_entries": 3
}
```

3.2 Batch Upload Questions

POST /api/questions/batch

Request Body:

```
{
  "questions": [
    {
      "content": "Question 1 content",
      "options": { "A": "Opt A", "B": "Opt B", "C": "Opt C", "D": "Opt D"
    },
    "correct_answer": "A",
    "topic_id": "topic-uuid-1",
    "difficulty": 0.5,
    "discrimination": 1.0,
    "guessing": 0.25,
    "attributes": [
      { "attribute_id": "attr-uuid-1", "value": true }
    ]
  },
  {

```

```
    "content": "Question 2 content",
    "options": { "A": "Opt A", "B": "Opt B", "C": "Opt C", "D": "Opt D"
  },
  "correct_answer": "B",
  "topic_id": "topic-uuid-1",
  "difficulty": 0.7,
  "discrimination": 1.2,
  "guessing": 0.25
}
]
```

Response:

```
{
  "questions": [...],
  "q_matrix_entries_count": 5
}
```

3.3 Upload Question Image

POST /api/questions/{question_id}/image

Request: multipart/form-data

Form Fields:

- **image:** File (PNG, JPG, etc.)

Example (JavaScript):

```
const formData = new FormData();
formData.append('image', imageFile);

const response = await fetch(`/api/questions/${questionId}/image`, {
  method: 'POST',
  body: formData
});

const result = await response.json();
// result.url contains the image URL
```

Response:

```
{
  "success": true,
  "url": "https://supabase-url/storage/v1/object/public/question-
images/question-uuid-1/image.png",
  "file_name": "image.png",
  "file_size": 245678
}
```

3.4 Upload Option Images

Single Option Image:

```
POST /api/questions/{question_id}/options/{option_key}/image
```

Multiple Option Images:

```
POST /api/questions/{question_id}/options/images
```

Request: `multipart/form-data`

Form Fields:

- `option_A`: File
- `option_B`: File
- `option_C`: File
- `option_D`: File

Response:

```
{
  "success": true,
  "results": [
    {
      "option_key": "A",
      "success": true,
      "url": "https://....option_A.png"
    },
    {
      "option_key": "B",
      "success": true,
      "url": "https://....option_B.png"
    }
  ]
}
```

3.5 Edit Question

```
PUT /api/questions/{question_id}
```

Note: Use GET first to retrieve the question, then update

```
GET /api/questions/{question_id}
```

Response includes:

```
{
  "id": "question-uuid-1",
  "content": "Question text",
  "options": {...},
  "correct_answer": "A",
  "difficulty": 0.5,
  "attributes": [
    {
      "id": "attr-uuid-1",
      "name": "Understanding Newton's Laws",
      "value": true
    }
  ],
  "q_matrix": [0, 1, 3]
}
```

3.6 Delete Question

Soft Delete:

```
DELETE /api/questions/{question_id}
```

Permanent Delete:

```
DELETE /api/questions/{question_id}/permanent
```

3.7 Get Attributes for Topic

```
GET /api/topic/{topic_id}/attributes
```

Response:

```
[
  {
    "id": "attr-uuid-1",
    "name": "Understanding Newton's First Law",
    "description": "Ability to explain law of inertia",
    "topic_id": "topic-uuid-1"
  }
]
```

3.8 Create Attributes for Topic

POST /api/topic/{topic_id}/attributes

Request Body:

```
[
  {
    "name": "Understanding Newton's Laws",
    "description": "Comprehensive understanding of all three laws"
  },
  {
    "name": "Applying Newton's Laws",
    "description": "Ability to apply laws to real-world problems"
  }
]
```

Frontend Implementation Example

```
// Question Upload Component
interface QuestionFormData {
  content: string;
  options: { [key: string]: string };
  correct_answer: string;
  topic_id: string;
  difficulty: number;
  discrimination: number;
  guessing: number;
  questionImage?: File;
  optionImages?: { [key: string]: File };
  selectedAttributes: string[];
}

// Upload question with images
async function uploadQuestion(formData: QuestionFormData) {
```

```
// 1. Create question
const questionResponse = await fetch('/api/questions/create-with-attributes', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    question: {
      content: formData.content,
      options: formData.options,
      correct_answer: formData.correct_answer,
      topic_id: formData.topic_id,
      difficulty: formData.difficulty,
      discrimination: formData.discrimination,
      guessing: formData.guessing
    },
    selected_attributes: formData.selectedAttributes.map(id => ({
      attribute_id: id,
      value: true
    })))
  })
});

const { question } = await questionResponse.json();
const questionId = question.id;

// 2. Upload question image if exists
if (formData.questionImage) {
  const imageFormData = new FormData();
  imageFormData.append('image', formData.questionImage);

  await fetch(`/api/questions/${questionId}/image`, {
    method: 'POST',
    body: imageFormData
  });
}

// 3. Upload option images if exist
if (formData.optionImages && Object.keys(formData.optionImages).length > 0) {
  const optionsFormData = new FormData();
  Object.entries(formData.optionImages).forEach([key, file] => {
    optionsFormData.append(`option_${key}`, file);
  });

  await fetch(`/api/questions/${questionId}/options/images`, {
    method: 'POST',
    body: optionsFormData
  });
}

return questionId;
}

// Batch upload questions
```

```
async function batchUploadQuestions(questions: QuestionFormData[]) {
  const response = await fetch('/api/questions/batch', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      questions: questions.map(q => ({
        content: q.content,
        options: q.options,
        correct_answer: q.correct_answer,
        topic_id: q.topic_id,
        difficulty: q.difficulty,
        discrimination: q.discrimination,
        guessing: q.guessing,
        attributes: q.selectedAttributes.map(id => ({
          attribute_id: id,
          value: true
        })))
      })))
    })
  });

  return response.json();
}
```

4. PYQ Upload & Session Builder

Page: Previous Year Question Upload

Purpose: Allow content teams to ingest previous-year questions (PYQ) with metadata and instantly surface them for hierarchy browsing or practice sessions.

UX Flow

```
Select hierarchy (Exam → Subject → Chapter → Topic)
↓
Choose ingest mode (Single | Bulk JSON | Excel import)
↓
Enter PYQ content, options, correct answer
↓
Fill metadata (year, exam session, marks, source, tags, difficulty level,
question type)
↓
Attach topic attributes (reuse or create)
↓
Preview + upload
↓
Optional: launch a focused practice session
```

4.1 Upload Single PYQ

```
POST /api/pyq/upload/single
Content-Type: application/json

{
  "content": "What is the value of g?",
  "options": ["9.8 m/s^2", "9 m/s^2", "8 m/s^2", "10 m/s^2"],
  "correct_answer": "9.8 m/s^2",
  "exam_id": "exam-uuid",
  "subject_id": "subject-uuid",
  "chapter_id": "chapter-uuid",
  "topic_id": "topic-uuid",
  "metadata": {
    "year": 2024,
    "exam_session": "January",
    "marks_allocated": 2,
    "time_allocated": 120,
    "source": "Official Paper",
    "tags": ["modern-physics", "gravitation"],
    "difficulty_level": "Medium",
    "question_type": "MCQ"
  },
  "attributes": [
    { "attribute_id": "attr-uuid", "value": true }
  ]
}
```

Frontend Tip: Pre-fill `exam_id/subject_id/topic_id` from the current hierarchy selection so content authors only supply metadata.

4.2 Bulk Upload PYQ

```
POST /api/pyq/upload/bulk
Content-Type: application/json

{
  "questions": [ { ...single-pyq-payload... }, { ... } ]
}
```

- Validate each payload on the client before calling the API.
- Show a progress indicator and collate failures using `errors` array from the response.

4.3 Upload PYQ from Excel

```
POST /api/pyq/upload/excel
Content-Type: multipart/form-data

file: pyq_upload.xlsx
```


Column	Description	Required
content	Question stem	✓
options	JSON array or separated	✓
correct_answer	Correct option	✓
year	Numeric year	✓
exam_session	Session identifier	
paper_code	Paper reference	
question_number	Original numbering	
marks_allocated	Numeric mark	
time_allocated	Seconds/minutes for exam	
source	Book / institute / official	
tags	Comma-separated keywords	
difficulty_level	Easy/Medium/Hard	
question_type	MCQ/Numerical/Subjective	
exam_id ... concept_id	Hierarchy mapping	

Provide authors with a downloadable template: [GET /api/pyq/template/download](#).

4.4 Fetch PYQ Filter Options

[GET /api/pyq/filters/options](#)

Response structure supplies drop-down values for year, session, source, difficulty, and type.

4.5 Search PYQ by Hierarchy/Metadata

[GET /api/pyq/search?topic_id={topicId}&year=2024&source=Official&page_size=50](#)

- Supports hierarchy params ([exam_id](#), [subject_id](#), etc.) and metadata filters.
- Use this endpoint to populate “PYQ” tab in the question browser.

4.6 Create Practice Session from Filtered PYQ

[POST /api/pyq/session/create](#)
Content-Type: application/json

```
{
  "user_id": "admin-preview",
  "session_name": "Topic Drill – Gravitation",
  "filters": {
    "topic_id": "topic-uuid",
    "year": 2024,
    "difficulty_level": "Medium"
  },
  "time_limit": 30
}
```

Store the returned `session_id` if you want to offer a “Preview Session” link inside the admin UI.

Frontend Implementation Example

```
async function uploadSinglePYQ(form: PYQFormState) {
  const payload = buildSinglePayload(form);
  const response = await fetch('/api/pyq/upload/single', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload)
  });
  if (!response.ok) throw await response.json();
  const result = await response.json();
  return result;
}

function PYQUploadPage() {
  const [mode, setMode] = useState<'single' | 'bulk' | 'excel'>('single');
  const [metadataOptions, setMetadataOptions] = useState<PYQFilterOptions>
    ();

  useEffect(() => {
    fetch('/api/pyq/filters/options').then(r =>
      r.json()).then(setMetadataOptions);
  }, []);

  return (
    <Layout>
      <HierarchyNavigator />
      <ModeTabs value={mode} onChange={setMode} />
      {mode === 'single' && <SinglePYQForm onSubmit={uploadSinglePYQ}
options={metadataOptions} />}
      {mode === 'bulk' && <BulkPYQUpload />}
      {mode === 'excel' && <ExcelPYQUpload />}
      <RecentPYQUploadsTable />
    </Layout>
  );
}
```

5. Resource Upload Service

Page: Topic Resources

Purpose: Upload learning resources (videos, PDFs, virtual labs, 3D models, etc.) for topics

5.1 Add Single Resource to Topic

```
POST /api/topics/{topic_id}/resources
```

Request Body:

```
{
  "resource_type": "video",
  "title": "Introduction to Newton's Laws",
  "description": "Comprehensive video explaining all three laws",
  "url": "https://www.youtube.com/watch?v=example",
  "thumbnail_url": "https://img.youtube.com/vi/example/maxresdefault.jpg",
  "duration": 600,
  "metadata": {
    "platform": "YouTube",
    "quality": "1080p",
    "language": "English",
    "creator": "Khan Academy"
  },
  "order_index": 0
}
```

Resource Types:

- **video** - Video content (YouTube, Vimeo)
- **image** - Images and diagrams
- **3d_model** - 3D models (GLTF, OBJ)
- **animation** - Animated content
- **virtual_lab** - Virtual lab simulations
- **pdf** - PDF documents
- **interactive** - Interactive widgets
- **article** - Articles and blog posts
- **simulation** - Interactive simulations

Response:

```
{
  "id": "resource-uuid-1",
  "topic_id": "topic-uuid-1",
  "resource_type": "video",
  "title": "Introduction to Newton's Laws",
```

```
{
  "url": "https://www.youtube.com/watch?v=example",
  "duration": 600,
  "created_at": "2025-01-01T00:00:00Z"
}
```

5.2 Bulk Add Resources

POST /api/topics/{topic_id}/resources/bulk

Request Body:

```
{
  "resources": [
    {
      "resource_type": "video",
      "title": "Lecture 1: Introduction",
      "url": "https://youtube.com/v1",
      "duration": 600
    },
    {
      "resource_type": "pdf",
      "title": "Study Guide",
      "url": "https://example.com/guide.pdf",
      "file_size": 2048000
    },
    {
      "resource_type": "virtual_lab",
      "title": "Physics Simulator",
      "url": "https://phet.colorado.edu/physics",
      "metadata": {
        "platform": "PhET",
        "interactive": true
      }
    }
  ]
}
```

Response:

```
{
  "success": true,
  "created_count": 3,
  "resources": [...]
}
```

5.3 Get Resources for Topic

```
GET /api/topics/{topic_id}/resources
```

Query Parameters:

- `resource_type` (optional): Filter by type
- `is_active` (optional): true/false (default: true)

Response:

```
{
  "topic_id": "topic-uuid-1",
  "topic_name": "Newton's Laws of Motion",
  "resources": [
    {
      "id": "resource-uuid-1",
      "resource_type": "video",
      "title": "Introduction to Newton's Laws",
      "url": "https://youtube.com/...",
      "thumbnail_url": "https://...",
      "duration": 600,
      "order_index": 0,
      "is_active": true
    }
  ],
  "count": 1
}
```

5.4 Update Resource

```
PUT /api/resources/{resource_id}
```

Request Body:

```
{
  "title": "Updated Title",
  "description": "Updated description",
  "metadata": {
    "quality": "4K",
    "updated": true
  }
}
```

5.5 Delete Resource

Soft Delete:

```
DELETE /api/resources/{resource_id}
```

Permanent Delete:

```
DELETE /api/resources/{resource_id}/permanent
```

5.6 Get Chapter Resources (All Topics)

```
GET /api/chapters/{chapter_id}/resources
```

Response:

```
{
  "chapter_id": "chapter-uuid-1",
  "chapter_name": "Mechanics",
  "topics": [
    {
      "topic_id": "topic-uuid-1",
      "topic_name": "Newton's Laws",
      "resources": [...],
      "resource_count": 5
    }
  ],
  "total_resources": 5
}
```

Frontend Implementation Example

```
// Resource Upload Component
interface ResourceFormData {
  resource_type: string;
  title: string;
  description: string;
  url: string;
  thumbnail_url?: string;
  duration?: number;
  file_size?: number;
  metadata?: any;
}

// Upload single resource
```

```

async function uploadResource(topicId: string, data: ResourceFormData) {
  const response = await fetch(`/api/topics/${topicId}/resources`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  });
  return response.json();
}

// Bulk upload resources
async function bulkUploadResources(topicId: string, resources:
ResourceFormData[]) {
  const response = await fetch(`/api/topics/${topicId}/resources/bulk`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ resources })
  });
  return response.json();
}

// Get resources with filtering
async function getTopicResources(topicId: string, type?: string) {
  let url = `/api/topics/${topicId}/resources`;
  if (type) url += `?resource_type=${type}`;

  const response = await fetch(url);
  return response.json();
}

// Resource type selector
const RESOURCE_TYPES = [
  { value: 'video', label: 'Video', icon: '🎥' },
  { value: '3d_model', label: '3D Model', icon: '🖨️' },
  { value: 'virtual_lab', label: 'Virtual Lab', icon: '🧪' },
  { value: 'pdf', label: 'PDF Document', icon: '📄' },
  { value: 'interactive', label: 'Interactive', icon: '🎮' },
  { value: 'article', label: 'Article', icon: '📰' },
  { value: 'simulation', label: 'Simulation', icon: '⚙️' },
  { value: 'animation', label: 'Animation', icon: '✨' },
  { value: 'image', label: 'Image', icon: '🖼️' }
];

```

6. Enhanced Question Fetching

Page: Question Browser

Purpose: Fetch and display questions with Q-matrix at any hierarchy level

6.1 Get Questions with Enhanced Attributes

```
GET /api/hierarchy/{level}/{item_id}/questions/enhanced
```

Parameters:

- **level**: exam, subject, chapter, topic, class
- **item_id**: UUID of the item
- **page** (optional): Page number (default: 1)
- **page_size** (optional): Items per page (default: 20)

Response:

```
{
  "level": "topic",
  "level_id": "topic-uuid-1",
  "total_questions": 150,
  "attributes": [
    {
      "id": "attr-uuid-1",
      "name": "Understanding Newton's First Law",
      "description": "...",
      "topic_id": "topic-uuid-1"
    },
    {
      "id": "attr-uuid-2",
      "name": "Applying Newton's First Law",
      "description": "...",
      "topic_id": "topic-uuid-1"
    }
  ],
  "attribute_count": 2,
  "questions": [
    {
      "id": "question-uuid-1",
      "content": "What is Newton's First Law?",
      "options": {...},
      "correct_answer": "A",
      "difficulty": 0.5,
      "discrimination": 1.2,
      "guessing": 0.25,
      "q_vector": [1, 1, 0, 0],
      "attribute_count": 2,
      "question_image_url": "https://...",
      "option_images": {
        "A": "https://...",
        "B": "https://..."
      }
    }
  ],
  "pagination": {
    "page": 1,
```



```
    "page_size": 20,  
    "total": 150,  
    "total_pages": 8,  
    "has_more": true  
  }  
}
```

Q-Vector Explanation:

- Binary vector where 1 means the question tests that attribute
- Index corresponds to position in `attributes` array
- Example: `[1, 1, 0, 0]` means question tests first two attributes

6.2 Get Questions with Hierarchy Details

```
GET /api/hierarchy/{level}/{item_id}/questions
```

Parameters:

- `page`: Page number
- `page_size`: Items per page

Response:

```
{  
  "data": [  
    {  
      "id": "question-uuid-1",  
      "content": "Question text",  
      "options": {...},  
      "correct_answer": "A",  
      "attributes": [  
        {  
          "id": "attr-uuid-1",  
          "name": "Understanding Newton's Laws",  
          "description": "...",  
          "value": true  
        }  
      ],  
      "q_matrix": [0, 2, 5]  
    }  
  ],  
  "pagination": {  
    "total": 150,  
    "page": 1,  
    "page_size": 20,  
    "total_pages": 8,  
    "has_more": true  
  }  
}
```

```
}  
}
```

6.3 Search Questions

GET /api/search/questions

Query Parameters:

- `exam_id` (optional)
- `subject_id` (optional)
- `chapter_id` (optional)
- `topic_id` (optional)
- `concept_id` (optional)
- `text_search` (optional): Search in question content
- `difficulty_min` (optional): Minimum difficulty
- `difficulty_max` (optional): Maximum difficulty
- `page` (optional)
- `page_size` (optional)

Example:

GET /api/search/questions?topic_id=topic-uuid-1&difficulty_min=0.5&difficulty_max=0.8&page=1&page_size=20

Response:

```
{  
  "data": [...],  
  "pagination": {...}  
}
```

6.4 Get Item Bank for Level

GET /api/item-bank/{level}/{item_id}

Parameters:

- `page, page_size`: Pagination
- `difficulty_min, difficulty_max`: Filter by difficulty
- `text_search`: Search query

Response:

```
{
  "data": [
    {
      "id": "question-uuid-1",
      "content": "Question text",
      "difficulty": 0.5,
      "discrimination": 1.2,
      "attributes": [...]
    }
  ],
  "pagination": {...}
}
```

Frontend Implementation Example

```
// Question Browser Component
interface QuestionBrowserState {
  level: 'exam' | 'subject' | 'chapter' | 'topic';
  itemId: string;
  questions: EnhancedQuestion[];
  attributes: Attribute[];
  pagination: PaginationInfo;
  filters: QuestionFilters;
}

interface EnhancedQuestion {
  id: string;
  content: string;
  options: { [key: string]: string };
  correct_answer: string;
  difficulty: number;
  q_vector: number[];
  attribute_count: number;
  question_image_url?: string;
  option_images?: { [key: string]: string };
}

// Fetch questions with Q-matrix
async function fetchEnhancedQuestions(
  level: string,
  itemId: string,
  page: number = 1
): Promise<EnhancedQuestionResponse> {
  const response = await fetch(
    `/api/hierarchy/${level}/${itemId}/questions/enhanced?
page=${page}&page_size=20`
  );
  return response.json();
}
```

```

}

// Display Q-matrix as visual grid
const QMatrixDisplay: React.FC<{ qVector: number[], attributes:
Attribute[] }> =
  ({ qVector, attributes }) => {
    return (
      <div className="q-matrix-grid">
        {attributes.map((attr, index) => (
          <div
            key={attr.id}
            className={`q-matrix-cell ${qVector[index] ? 'active' :
'inactive'}`}
            title={attr.name}
          >
            {qVector[index] ? '✓' : '○'}
          </div>
        ))}
      </div>
    );
  };

// Question card with Q-matrix
const QuestionCard: React.FC<{ question: EnhancedQuestion, attributes:
Attribute[] }> =
  ({ question, attributes }) => {
    return (
      <div className="question-card">
        <div className="question-content">
          {question.question_image_url && (
            <img src={question.question_image_url} alt="Question" />
          )}
          <p>{question.content}</p>
        </div>

        <div className="question-options">
          {Object.entries(question.options).map(([key, value]) => (
            <div key={key} className="option">
              {question.option_images?.[key] && (
                <img src={question.option_images[key]} alt={`Option
${key}`} />
              )}
              <span>{key}. {value}</span>
              {question.correct_answer === key && <span
className="correct">✓</span>}
            </div>
          ))}
        </div>

        <div className="question-metadata">
          <div>Difficulty: {question.difficulty.toFixed(2)}</div>
          <div>Tests {question.attribute_count} attributes</div>
        </div>
      </div>
    );
  };

```

```

        <div className="q-matrix-section">
          <h4>Q-Matrix (Attributes Tested)</h4>
          <QMatrixDisplay qVector={question.q_vector} attributes=
{attributes} />
          <div className="attribute-list">
            {attributes.map((attr, index) => (
              question.q_vector[index] === 1 && (
                <span key={attr.id} className="attribute-tag">
                  {attr.name}
                </span>
              )
            ))}
          </div>
        </div>
      </div>
    );
  };

// Filter panel
const QuestionFilters: React.FC<{ onFilterChange: (filters: any) => void
}> =
  ({ onFilterChange }) => {
    return (
      <div className="filters">
        <input
          type="text"
          placeholder="Search questions..."
          onChange={(e) => onFilterChange({ text_search: e.target.value
        }}}
      />
      <div className="difficulty-range">
        <label>Difficulty Range:</label>
        <input type="number" min="0" max="3" step="0.1"
placeholder="Min" />
        <input type="number" min="0" max="3" step="0.1"
placeholder="Max" />
      </div>
    </div>
    );
  };

```

7. Complete API Reference

Hierarchy Management

Method	Endpoint	Purpose
GET	/api/hierarchy/exams	Get all exams
GET	/api/hierarchy/classes?exam_id={id}	Get classes for school exam
GET	/api/hierarchy/subjects?exam_id={id}	Get subjects for competitive exam

Method	Endpoint	Purpose
GET	/api/hierarchy/subjects?class_id={id}	Get subjects for school exam class
GET	/api/hierarchy/chapters?subject_id={id}	Get chapters for subject
GET	/api/hierarchy/topics?chapter_id={id}	Get topics for chapter
GET	/api/hierarchy/concepts?topic_id={id}	Get concepts for topic
GET	/api/hierarchy/tree	Get complete hierarchy tree
POST	/api/exams	Create exam (competitive or school)
POST	/api/classes	Create class (school exams only)
POST	/api/subjects	Create subject (provide exam_id or class_id)
POST	/api/chapters	Create chapter
POST	/api/topics	Create topic
POST	/api/concepts	Create concept

Statistics & Counts

Method	Endpoint	Purpose
GET	/api/hierarchy/exam/{id}/question-count	Get question count for exam
GET	/api/hierarchy/subject/{id}/question-count	Get question count for subject
GET	/api/hierarchy/chapter/{id}/question-count	Get question count for chapter
GET	/api/hierarchy/topic/{id}/question-count	Get question count for topic
GET	/api/hierarchy/{level}/{id}/stats	Get detailed statistics

Question Management

Method	Endpoint	Purpose
POST	/api/questions/create-with-attributes	Create question with attributes
POST	/api/questions/batch	Batch create questions
GET	/api/questions/{id}	Get specific question
PUT	/api/questions/{id}	Update question

Method	Endpoint	Purpose
DELETE	/api/questions/{id}	Soft delete question
GET	/api/hierarchy/{level}/{id}/questions	Get questions for hierarchy level
GET	/api/hierarchy/{level}/{id}/questions/enhanced	Get questions with Q-matrix
GET	/api/search/questions	Search questions

Image Upload

Method	Endpoint	Purpose
POST	/api/questions/{id}/image	Upload question image
POST	/api/questions/{id}/options/{key}/image	Upload single option image
POST	/api/questions/{id}/options/images	Upload multiple option images
DELETE	/api/questions/{id}/images	Delete all question images

Attributes

Method	Endpoint	Purpose
GET	/api/topic/{id}/attributes	Get attributes for topic
POST	/api/topic/{id}/attributes	Create attributes for topic
PUT	/api/attributes/{id}	Update attribute
DELETE	/api/attributes/{id}	Delete attribute

Resources

Method	Endpoint	Purpose
POST	/api/topics/{id}/resources	Add resource to topic
POST	/api/topics/{id}/resources/bulk	Bulk add resources
GET	/api/topics/{id}/resources	Get topic resources
GET	/api/chapters/{id}/resources	Get chapter resources (grouped)
GET	/api/resources/{id}	Get specific resource
PUT	/api/resources/{id}	Update resource
DELETE	/api/resources/{id}	Soft delete resource
DELETE	/api/resources/{id}/permanent	Permanently delete resource

UI/UX Recommendations

1. Dashboard Page

Admin Dashboard

[User ▼]

Exams

15

Questions

12,450

Resources

340

Users

250

Filter by:

[Exam ▼] [Subject ▼] [Chapter ▼] [Topic ▼]

Statistics Chart

Question Distribution by Difficulty

2. Hierarchy Creation Page

Competitive Exam Path:

Create Content

[Dashboard]

1. Select Exam Type

●

 Competitive Exam

○

 School Exam

2. Hierarchical Selection (Competitive Path)

Exam: [JEE Main 2025 ▼] [+ New]

Subject: [Physics ▼] [+ New]

Chapter: [Mechanics ▼] [+ New]

Topic: [Newton's Laws ▼] [+ New]

Concept: [First Law of Motion ▼] [+ New]

Breadcrumb: JEE Main > Physics > Mechanics > ...

3. Create New Item

Level: [Topic ▼]

Name: [_____]Description: [_____]

[Cancel] [Create]

40 / 46

School Exam Path:

Create Content

[Dashboard]

1. Select Exam Type

○ Competitive Exam

● School Exam

2. Hierarchical Selection (School Path)

Exam: [CBSE Board ▼] [+ New]

Class: [Class 10 ▼] [+ New]

Subject: [Physics ▼] [+ New]

Chapter: [Light ▼] [+ New]

Topic: [Reflection & Refraction ▼] [+ New]

Concept: [Laws of Reflection ▼] [+ New]

← EXTRA LEVEL

Breadcrumb: CBSE > Class 10 > Physics > Light > ...

3. Create New Item

Level: [Class ▼]

Name: [Class 11]

Class Number: [11]

Section: [Science]

[Cancel]

[Create]

3. Question Upload Page

4. PYQ Upload Page

- Keep question form side-by-side with metadata to minimise scrolling.
- Prefill hierarchy fields from the left navigation breadcrumbs.
- Offer inline validation for duplicate year/paper/question combinations before upload.
- Provide a sticky summary panel showing how many PYQ were uploaded in the current session and quick actions to launch practice sessions.

Upload Questions

[Back] [Save]


Navigate to Topic:





JEE Main > Physics > Mechanics > Newton's Laws

Upload Mode: ○ Single ● Batch

41 / 46

Question 1

Question Text:
[_____]
[Upload Image] 

Options:
A: [_____]  ☐ Correct
B: [_____]  ☒ Correct
C: [_____]  ☐ Correct
D: [_____]  ☐ Correct

3PL Parameters:
Difficulty: [0.5] (-3 to 3)
Discrimination: [1.2] (0 to 3)
Guessing: [0.25] (0 to 1)

Attributes:
☒ Understanding Newton's Laws
☒ Applying Forces
☐ Problem Solving
[+ Create New Attribute]

[+ Add Another Question]

[Upload Batch] [Cancel]

5. Resource Upload Page

Add Learning Resources

[Back] [Save]

Topic: Newton's Laws of Motion

Resource Type: [Video ▼]

Title: [_____]

Description: [_____] [_____]

URL: [https://youtube.com/watch?v=_____]

Thumbnail URL: [https://..._____]

Duration (seconds): [600]

Additional Info (Metadata):


Platform: [YouTube]


Quality: [1080p]


Language: [English]


[Cancel] [Add Resource]

Existing Resources

 Introduction Video (10:00)

 Study Guide PDF

 Virtual Lab Simulation

 3D Model of Forces

6. Question Browser Page

Question Browser [Edit] [Delete]

Filters: [Search...] [Difficulty: All ▼]

Level: Topic > Newton's Laws of Motion

Total Questions: 150 | Page 1 of 8

Attributes (5): Understanding Laws | Applying Forces | Problem Solving

Question #1

[Image]

What is Newton's First Law of Motion?

A. Law of Inertia ✓

B. $F = ma$

C. Action-Reaction

D. None of the above

Difficulty: 0.5 | Discrimination: 1.2

Q-Matrix: [✓] [✓] [] [] []

Tests: Understanding Laws, Applying Forces

[Edit] [Delete] [View Details]

Question #2

...

[< Previous] [1] [2] [3] ... [8] [Next >]

43 / 46

Error Handling

Standard Error Response Format

```
{
  "error": "Error message here",
  "details": "Additional details if available"
}
```

Common HTTP Status Codes

- 200 - Success
- 201 - Created
- 400 - Bad Request (validation error)
- 404 - Not Found
- 500 - Internal Server Error

Error Handling Example

```
async function apiRequest(url: string, options?: RequestInit) {
  try {
    const response = await fetch(url, options);

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

    return response.json();
  } catch (error) {
    console.error('API Error:', error);
    // Show user-friendly error message
    showErrorToast(error.message);
    throw error;
  }
}
```

Authentication (If Required)

If your backend requires authentication, add headers:

```
const API_TOKEN = 'your-auth-token';

fetch('/api/endpoint', {
  headers: {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${API_TOKEN}`
  }
});
```

Testing Endpoints

Use the provided Jupyter notebook for testing:

- [topic_resources_test.ipynb](#) - Resource API tests
- [endpoint_coverage_test.ipynb](#) - Complete API tests

Or use cURL:

```
# Get exams
curl http://localhost:5200/api/hierarchy/exams

# Create exam
curl -X POST http://localhost:5200/api/exams \
  -H "Content-Type: application/json" \
  -d '{"name":"JEE Main","description":"Engineering exam","exam_type":"competitive"}'

# Upload question
curl -X POST http://localhost:5200/api/questions/create-with-attributes \
  -H "Content-Type: application/json" \
  -d '{"question":{...},"selected_attributes":[...]}'
```

Additional Resources

- **API Documentation:** [/docs/TOPIC_RESOURCES_API.md](#)
- **Setup Guide:** [/TOPIC_RESOURCES_SETUP.md](#)
- **Database Schema:** [/CREATE_TOPIC_RESOURCES_TABLE.sql](#)

Support

For questions or issues:

1. Check the test notebooks for working examples
2. Review the API documentation
3. Verify server is running on port 5200

4. Check Supabase table structure matches schema

Server Start Command:

```
python -m app.main
```

Default Port: 5200 **Base URL:** <http://localhost:5200/api>