DWIJ Platform – User Context Layer & Syllabus Tree System

Project Objective

To build a **flexible, intelligent, and interactive syllabus tree system** that supports multiple examinations (e.g., CUET, CAT, SSC CGL), adapts to each exam's unique structure, and tracks a user's learning progress in real time.

This system will:

- Support **dynamic syllabus tree creation per exam**
- Allow **per-user performance tracking per syllabus node**
- Power **adaptive test generation** and personalized learning
- Provide **interactive frontend visualizations** like Duolingo
- Include an **admin interface** to define and edit trees

Core Architecture Overview

The system is built on **two tightly integrated layers**:

1. Syllabus Graph Layer

- Represents the hierarchical structure of an exam syllabus.
- Each node (subject, topic, etc.) is a `SyllabusNode`.

2. User Context Layer

- Overlays per-user performance data (confidence, accuracy, etc.) on the syllabus tree.
- This data is stored in `UserSyllabusNodeStats`.

Why Two Layers?

Separating the **syllabus structure** from the **user's performance data** offers:

- **Modularity**: One syllabus → many users
- **Flexibility**: New exams can use the same tree model
- **Efficiency**: Tree can be cached; user stats queried independently
- **Visual Adaptability**: Tree nodes can be colored, sized, or styled per-user

1. Database Schemas (MongoDB)

A. `SyllabusNode` – (The Tree Node)

Represents a single node in the syllabus hierarchy. This node can be a subject, topic, subtopic, or even micro-skill, depending on the exam structure.

🔧 **Structure:**

```
SyllabusNode {
  _id: ObjectId;
  examCode: string;          // e.g. "CUET", "CAT", "SSC_CGL"
  name: string;              // Node name: "Arithmetic", "Grammar"
  type: string;              // e.g. "subject", "topic", "subtopic"
  description?: string;      // Optional explanation
  parentId: ObjectId | null; // null for root nodes
  childrenIds: ObjectId[];   // Populated recursively
  displayOrder: number;      // Position among siblings
  estimatedTime?: number;    // In minutes (helpful for planners)
  weightageEstimate?: number;// Optional for exams with topic-wise
weight
  isTerminal: boolean;       // Marks leaf node for question linkage
  tags?: string[];           // E.g. ["math", "quant", "grammar"]
  linkedQuestionIds: ObjectId[];// Used for targeted test generation
  createdAt: Date;
  updatedAt: Date;}
```

Why This Design?

- **Flexible for all exams** — “topic” in CUET might be “section” in CAT.
- **Allows tree traversal** — by using `parentId` + `childrenIds`.

- **Supports AI generation** — via `tags`, `estimatedTime`, etc.
- **Links directly to questions** — allowing precise test targeting.

B. `UserSyllabusNodeStats` – (The Performance Overlay)

Tracks how well a specific user is performing for a given syllabus node. This allows:

- Visual feedback in the tree
- Adaptive test generation
- Progress summaries

🔧 Structure:

```
UserSyllabusNodeStats {
  userId: ObjectId;
  syllabusNodeId: ObjectId;

  confidence: number;         // Range: 0.0 to 1.0
  accuracy: number;           // % over last N attempts
  lastAttempted: Date;
  decayRate: number;          // How quickly confidence decays
  avgTimeSpent: number;       // Average time per question in seconds
  attempts: number;           // Total attempts for this node
  correctAttempts: number;
  masteryStatus: "unseen" | "learning" | "mastered" | "struggling";
  streak: number;             // Daily streak of engagement
  updatedAt: Date;
}
```

Why This Design?

- Enables real-time **color-coded tree UI**.
- Stores all key metrics to track knowledge decay or growth.
- Allows backend to **prioritize nodes for revision/test**.

- Separates system data (tree) from user-specific data (performance).

2. Backend APIs (NestJS)

Admin APIs (for Syllabus Management)

Method	Endpoint	Purpose
`POST`	`/admin/syllabus-node`	Create a node
`PUT`	`/admin/syllabus-node/:id`	Edit node details
`DELETE`	`/admin/syllabus-node/:id`	Delete node
`GET`	`/admin/syllabus-tree?examCode=CUET`	Fetch entire tree (recursive)
`POST`	`/admin/syllabus-node/:id/add-question`	Attach questions to terminal node

Learner APIs (for User Context)

Method	Endpoint	Description
`GET`	`/user/syllabus-tree?examCode=CAT`	Returns tree with merged `UserSyllabusNodeStats`
`GET`	`/user/syllabus-node/:id/stats`	Stats for a specific node
`POST`	`/user/syllabus-node/:id/update-stats`	Updates performance after test or practice
`GET`	`/user/weakest-nodes?limit=5`	Returns weakest nodes by confidence/accuracy
`POST`	`/user/generate-test-from-node/:id`	Generates test from node & children
`GET`	`/user/syllabus-progress-summary`	Overall progress with graph data

3. Frontend Guidelines

Tree UI (User-Facing)

Features:

- Interactive collapsible tree
- Color-coded nodes based on `masteryStatus`:
  - 🟥 = “struggling”
  - 🟨 = “learning”
  - 🟩 = “mastered”
  - ⚪ = “unseen”
- Progress bar inside node
- Tooltip/hover showing confidence, accuracy, time
- Click: open detail modal + CTA: "Start Practice" or "Revise"

Libraries (Suggested):

- `react-d3-tree`
- `react-tree-graph`
- Custom SVG tree layout with Tailwind CSS

Admin Tree Builder

Features:

- Add/edit/delete nodes
- Drag-and-drop reordering
- Set type, name, description
- Assign questions (autocomplete)
- Preview as learner

- Save tree per examCode

4. Developer Deliverables Checklist

Task	Assigned To	Priority
Define `SyllabusNode` schema (MongoDB)	Backend	✅ High
Define `UserSyllabusNodeStats` schema	Backend	✅ High
Recursive tree fetch utility	Backend	✅ High
Create full admin CRUD APIs	Backend	✅ High
Create learner stats API	Backend	✅ High
Frontend tree rendering	Frontend	✅ High
Tree UI state based on mastery/confidence	Frontend	✅ Medium
Tree editor for admin	Frontend	✅ Medium
Connect test generation to node traversal	Backend	✅ Medium
Global syllabus progress widget	Frontend	✅ Low
Daily decay job	Backend	✅ Low

5. Utility Functions (Backend)

Tree Traversal Utility

```
function getAllDescendantNodeIds(rootNodeId: ObjectId): ObjectId[] {
  // BFS/DFS traversal to collect children recursively
}
```

Aggregate Node Stats (Optional)

```
function aggregateStatsFromChildren(children:
UserSyllabusNodeStats[]): AggregatedStats {
  // e.g. mean confidence, weighted accuracy
}
```

6. Visualization Ideas

- Show **progress ring** around each node (like Duolingo units)
- Use animated transitions when node updates (Framer Motion)
- Display **summary bar** on top: Total mastery %, active streak, time spent
- Allow filters: "Show only weakest", "Hide mastered", "Revision mode"

7. Security & Scalability

- Use role-based guards for `/admin` routes.
- Index `userId + syllabusNodeId` in `UserSyllabusNodeStats`.
- Use Redis for caching tree fetches.
- Consider `batch sync` for daily decay updates.