Mind-Map Creator:

Product overview:

empowers users to explore any topic in depth by building a hierarchical network of nodes (topics, subtopics, pages). Each node can host rich content—editable documents (Confluence-style), —enabling a flexible, tree-structured knowledge space.

Feature & Specification Overview

A concise, user-centric summary of what Exploration Space offers:

1. **Spaces & Topics:** Create distinct "Exploration Spaces" to organize any subject—each with a clear title and description.
2. **Hierarchical Navigation:** Intuitive tree structure: start at a main topic and drill down into nested subtopics and nodes.
3. **Rich Content Pages:** Within each node, author or edit pages using a simple, Confluence-style editor (text formatting, images, tables).
4. **Confluence-based Pages:** All nodes support rich pages via a unified Confluence-style editor—no separate uploads or embeds needed
5. **Drag-&-Drop Organization:** Easily reorder topics and subtopics by dragging nodes within the tree view.
6. **Search & Jump:** Global search to find nodes or pages; one-click navigation to any content in your space.
7. **User Management:** Secure signup/login; each user's spaces are private by default, with future ability to invite collaborators.
8. **Responsive Design:** Optimized for desktop browsers, with clear layouts and collapsible panels for efficient focus.

Technical Architecture:

Components

1. API Gateway(Fast API) : All the APIs should
   - Follow RESTful API standards.
   - Use correct HTTP status codes
   - Should have validations
   - Should have detailed documentation
     - API Specifications:
       - / auth route is public and is used for authentication(can use Cognito)
       - `/spaces`, `/nodes`, and `/pages` routes, each of them have authorization through JWT.
       - **Validation:**
         - Every request body is validated against a JSON Schema (or Zod schema) defined in API Gateway request validators.
         - On validation failure, return **400 Bad Request** with a structured error payload.
2. Lambda:
   a. Access to dynamoDB, S3 buckets.
   b. Lambda functions for each task of each endpoint is seperate
   c. functions are:
      i. `SpacesCreateHandler`

ii. `SpacesListHandler`
iii. `SpacesTreeHandler`
iv. `SpacesUpdateHandler`
v. `SpacesDeleteHandler`
vi. `NodesAddHandler`
vii. `NodesRenameHandler`
viii. `NodesDeleteHandler`
ix. `NodesReorderHandler`
x. `PagesCreateHandler`
xi. `PagesGetHandler`
xii. `PagesUpdateHandler`
xiii. `PagesListBySpaceHandler`

3. DynamoDB:
   a. Data storage Architecture:
   b. DynamoDB Schemas
      i. **Table: Spaces**
         - **Partition Key:** `PK = "SPACE#<spaceId>"`
         - **Sort Key:** `SK = "META"`
         - **Attributes:** name, description, ownerId, createdAt, updatedAt
      ii. **Table: Nodes**
         - **Partition Key:** `PK = "SPACE#<spaceId>"`
         - **Sort Key:** `SK = "NODE#<nodeId>"`
         - **Attributes:** parentNodeId, title, orderIndex, createdAt, updatedAt
      iii. **Table: Pages**
         - **Partition Key:** `PK = "NODE#<nodeId>"`
         - **Sort Key:** `SK = "PAGE#<pageId>"`
         - **Attributes:** s3Key (reference to S3 object), version, createdAt, updatedAt
4. S3 bucket:

Object Key Layout
Organized to support retrieval by Space → Node → Page, with separation for attachments.

Content Type	S3 Object Key Format	Description
Page HTML	`pages/<spaceId>/<nodeId>/<pageId>/content.html`	Stores full rendered HTML from editor
Attachments	`attachments/<spaceId>/<nodeId>/<pageId>/<attachmentId>.<ext>`	Images, documents, embedded files

If website hosting is to be done on S3 itself, then another S3 bucket is used for the hosting the website itself and using CloudFront for edge delivery for all these stuff.

Core Entities of the program

---

1. User

Represents an authenticated user of the system.

- **Attributes:**
  - `userId: UUID`
  - `email: string`
  - `passwordHash: string`
  - `roles: string[]`
- **Events:**
  - `UserRegistered`
  - `UserAuthenticated`
- **Responsibilities:**
  - Can own spaces
  - Can authenticate to access protected resources

---

2. Space

Top-level container for organizing knowledge.

- **Attributes:**
  - `spaceId: UUID`
  - `name: string`
  - `description: string`
  - `ownerId: UUID`
  - `createdAt: Date`
  - `updatedAt: Date`
- **Events:**
  - `SpaceCreated`
  - `SpaceRenamed`
  - `SpaceDeleted`
- **Responsibilities:**
  - Holds a hierarchical node structure
  - Owns pages and attachments

---

3. Node

Represents a subtopic or a branch within a space.

- **Attributes:**
  - `nodeId: UUID`
  - `spaceId: UUID`
  - `parentNodeId?: UUID`
  - `title: string`
  - `orderIndex: number`

- `createdAt: Date`
- `updatedAt: Date`

- **Events:**
  - `NodeAdded`
  - `NodeRenamed`
  - `NodeRemoved`
  - `NodeReordered`
- **Responsibilities:**
  - Maintains tree structure within a space
  - Each node can contain one or more pages

---

4. Page

Confluence-style content page under a node.

- **Attributes:**
  - `pageId: UUID`
  - `nodeId: UUID`
  - `spaceId: UUID` (derived from node)
  - `s3Key: string` (S3 path to the content)
  - `version: number`
  - `createdAt: Date`
  - `updatedAt: Date`
- **Events:**
  - `PageCreated`
  - `PageUpdated`
  - `PageDeleted`
- **Responsibilities:**
  - Stores HTML content in S3
  - Metadata resides in DynamoDB

API endpoints and detailed API specifications:

1. Authentication (via AWS Cognito)

1.1 Sign Up

POST /api/auth/signup

- **Purpose:** Register new user with Cognito User Pool
- **Headers:** Content-Type: application/json
- **Request JSON:**
  {
  "email": "user@example.com", // required, valid email
  "password": "P@ssw0rd!" // required, meets User-Pool password policy
  }

- **Success (201 Created):**
  {
  "userId": "cognito-sub-uuid",
  "email": "user@example.com",
  "userConfirmed": false, // false if email confirmation pending
  "createdAt": "2025-05-20T12:00:00Z"
  }
- **Errors:**
  • 400 if missing/invalid fields
  • 409 if email already registered

1.2 Confirm Sign Up

POST /api/auth/confirm

- **Purpose:** Confirm user via Cognito verification code
- **Request JSON:**
  {
  "email": "user@example.com", // required
  "confirmationCode": "123456" // required, from email/SMS
  }
- **Success (200 OK):**
  { "message": "User confirmed" }
- **Errors:** 400 | 404 | 409

1.3 Login (Authenticate)

POST /api/auth/login

- **Purpose:** Exchange credentials for JWT tokens
- **Request JSON:**
  {
  "email": "user@example.com",
  "password": "P@ssw0rd!"
  }
- **Success (200 OK):**
  {
  "accessToken": "ey...", // use this in Authorization header
  "refreshToken": "bey...", // to obtain new access tokens
  "expiresIn": 3600 // seconds until accessToken expires
  }
- **Errors:**
  • 400 invalid payload
  • 401 unauthorized (wrong password / user not confirmed)
- **Protected Endpoints require:**
  • Authorization: Bearer <Cognito-JWT-Access-Token>
  • Content-Type: application/json
- **Error Response:**
  { errorCode, message, details? }
  • errorCode ∈ {BadRequest, Unauthorized, Forbidden, NotFound, Conflict, InternalError}
- **Validation:** All request bodies validated via API Gateway JSON Schema

2. SPACES

All `/api/spaces` endpoints require Authorization header.

2.1 **Create Space**

- **POST /api/spaces**
- **Purpose:** Persist a new Exploration Space record
- **Lambda:** SpacesCreateHandler
- **Request:** { name: string (required), description: string (optional) }
- **Stores:** DynamoDB "Spaces" table item with PK=`SPACE#<spaceId>`, SK=`META`, plus attributes `name`, `description`, `ownerId`, `createdAt`, `updatedAt`
- **Response (201):** { spaceId: string, name: string, description: string, createdAt: ISO-timestamp }
- **Errors:** 400

2.2 **List Spaces**

- **GET /api/spaces**
- **Purpose:** Retrieve all spaces owned by the caller
- **Lambda:** SpacesListHandler
- **Does:** Query DynamoDB "Spaces" table by ownerId GSI or by scanning PK prefix
- **Response (200):** [{ spaceId, name, description, createdAt } ...]

2.3 **Get Space Tree**

- **GET /api/spaces/{spaceId}/tree**
- **Purpose:** Return full nested node structure for a space
- **Lambda:** SpacesTreeHandler
- **Does:** Query "Nodes" table for PK=`SPACE#<spaceId>`, then recursively assemble children lists
- **Response (200):** { spaceId, nodes: [{ nodeId, title, children: [...] } ...] }
- **Errors:** 404 (space not found or not owned)

2.4 **Rename Space**

- **PUT /api/spaces/{spaceId}**
- **Purpose:** Update space's name/description
- **Lambda:** SpacesUpdateHandler
- **Request:** { name: string (required), description: string (optional) }
- **Stores:** DynamoDB UpdateItem on "Spaces" item, sets new name, description, updatedAt
- **Response (200):** { spaceId, name, description, updatedAt }
- **Errors:** 400, 404

2.5 **Delete Space**

- **DELETE /api/spaces/{spaceId}**
- **Purpose:** Remove space and all descendant data
- **Lambda:** SpacesDeleteHandler
- **Does:** DynamoDB DeleteItem on space, cascades deletes on Nodes and Pages (via batch or event-driven subscriber), S3 cleanup in subscriber
- **Response (204 No Content)**
- **Errors:** 404

---

3. NODES

All `/api/nodes` and `/api/spaces/{spaceId}/nodes` require JWT.

3.1 **Add Node**

- **POST /api/spaces/{spaceId}/nodes**
- **Purpose:** Create a new node under a parent (or root)
- **Lambda:** NodesAddHandler
- **Request:** { parentNodeId: string (UUID), title: string }
- **Stores:** DynamoDB "Nodes" table item with PK= `SPACE#<spaceId>`, SK= `NODE#<nodeId>`, plus attributes parentNodeId, title, orderIndex, createdAt, updatedAt
- **Response (201):** { nodeId, parentNodeId, title, createdAt }
- **Errors:** 400, 404

3.2 **Rename Node**

- **PUT /api/nodes/{nodeId}**
- **Purpose:** Change node title
- **Lambda:** NodesRenameHandler
- **Request:** { title: string }
- **Stores:** UpdateItem on "Nodes" table, sets title, updatedAt
- **Response (200):** { nodeId, title, updatedAt }
- **Errors:** 400, 404

3.3 **Delete Node**

- **DELETE /api/nodes/{nodeId}**
- **Purpose:** Remove a node and all its descendants
- **Lambda:** NodesDeleteHandler
- **Does:** Deletes node item, cascades to child nodes & pages; triggers S3 subscribers to clean up content
- **Response (204)**
- **Errors:** 404

3.4 **Reorder Nodes**

- **POST /api/spaces/{spaceId}/nodes/reorder**
- **Purpose:** Update the `orderIndex` of sibling nodes under a parent
- **Lambda:** NodesReorderHandler
- **Request:** { parentNodeId: string, order: [<nodeId>, ...] }
- **Stores:** BatchWriteItem to update each node's orderIndex in "Nodes" table
- **Response (200):** { spaceId, parentNodeId, order: [...] }
- **Errors:** 400, 404

---

4. PAGES

All `/api/pages` and `/api/nodes/{nodeId}/pages` require JWT.

4.1 **Create Page**

- **POST /api/nodes/{nodeId}/pages**
- **Purpose:** Persist a new Confluence-style page
- **Lambda:** PagesCreateHandler

- **Request:** { contentHTML: string (HTML), version: integer }
- **Stores:**
  • Writes metadata to DynamoDB “Pages” table item with PK=`NODE#<nodeId>`, SK=`PAGE#<pageId>`, attributes s3Key, version, createdAt, updatedAt
  • Uploads contentHTML to S3 at key `pages/<spaceId>/<nodeId>/<pageId>/content.html`
- **Response (201):** { pageId, nodeId, version, createdAt }
- **Errors:** 400, 404

4.2 **Get Page**

- **GET /api/pages/{pageId}**
- **Purpose:** Retrieve page HTML and metadata
- **Lambda:** PagesGetHandler
- **Does:**
  • Query DynamoDB for the page’s metadata (s3Key)
  • GetObject from S3 for contentHTML
- **Response (200):** { pageId, contentHTML: string, version, updatedAt }
- **Errors:** 404

4.3 **Update Page**

- **PUT /api/pages/{pageId}**
- **Purpose:** Overwrite content or bump page version
- **Lambda:** PagesUpdateHandler
- **Request:** { contentHTML: string, version: integer }
- **Stores:**
  • UpdateItem on “Pages” table (version, updatedAt)
  • Upload new contentHTML to S3 under same key (versioned by bucket)
- **Response (200):** { pageId, version, updatedAt }
- **Errors:** 400 (e.g., stale version), 404

4.4 **List Pages by Space**

- **GET /api/spaces/{spaceId}/pages**
- **Purpose:** List all pages across every node in a space
- **Lambda:** PagesListBySpaceHandler
- **Does:** Query “Pages” table GSI1 (GSI1PK=`SPACE#<spaceId>`)
- **Response (200):** [{ pageId, nodeId, version, createdAt } ...]
- **Errors:** 404