**Features**: Identify the key features

* **User Authentication**: Register, login, and manage user profiles.
* **Courses & Study Materials**: Provide study resources, mock tests, practice questions.
* **Video Lessons**: Integrate video lessons or tutorials.
* **Discussion Forum**: Let students discuss exam-related topics.
* **Admin Panel**: To manage content, users, and monitor site activities.
* **Notifications**: To alert users about updates or exam tips.

**1. User Authentication: Register, Login, and Manage User Profiles**

**Purpose**:  
This feature ensures that users can securely sign up, log in, and manage their personal profiles. It’s vital for tracking individual progress and allowing users to access their personalized content, such as saved study materials, mock test scores, etc.

**How to Implement**:

* **Registration**: Users can sign up by providing essential details such as **username**, **email**, and **password**.
  + Use password hashing (e.g., **bcryptjs**) for security.
  + Save user details in the MongoDB database.
* **Login**: Allow users to log in using their **email** and **password**. You’ll verify the password using bcrypt.
  + If successful, issue a **JWT (JSON Web Token)** for authentication, which users can store in localStorage or cookies.
  + JWT will be included in subsequent requests to authenticate users.
* **Profile Management**: Users should have the option to update their profiles, including their username, email, and password.
  + You can create a profile page where users can view and edit their information.
  + You can also store additional data, such as their exam progress, completed courses, etc.

**2. Courses & Study Materials: Provide Study Resources, Mock Tests, and Practice Questions**

**Purpose**:  
This feature helps users prepare for the GPAT and NIPER exams by offering organized study materials. It can include theoretical content, practice questions, mock tests, and more.

**How to Implement**:

* **Courses**: You can organize content into different categories, such as:
  + **Subjects**: e.g., Pharmacology, Pharmaceutical Chemistry, Pharmaceutics, etc.
  + **Topics**: Break down the subject into subtopics.
  + Users can browse through available courses and view detailed descriptions and resources.
* **Study Materials**: Upload PDFs, text-based resources, or links to external resources that students can access.
  + Store these materials in a **MongoDB database** and display them in a structured manner on the frontend.
* **Mock Tests & Practice Questions**:
  + Create a **Mock Test** section where users can take timed, simulated exams.
  + The mock tests should cover various topics and have questions of different difficulty levels.
  + Allow users to review their results after they complete a mock test, with answers and explanations.

**3. Video Lessons: Integrate Video Lessons or Tutorials**

**Purpose**:  
Video lessons help students understand difficult concepts in a more interactive manner. You can embed video tutorials that explain concepts, provide exam tips, or review practice questions.

**How to Implement**:

* **Video Hosting**: You can upload videos to platforms like **YouTube**, **Vimeo**, or host them directly on your website.
  + If you’re hosting them on YouTube or Vimeo, embed the video on your site using their provided embed codes.
  + If you want to host videos yourself, you could use **AWS S3**, **Cloudinary**, or similar services to store and stream videos.
* **Course Integration**: Each course or topic can have video lessons related to it. When users click on a specific topic, they can watch the corresponding video.
* **Interactive Elements**: Add features like **pause**, **rewind**, and **speed adjustment** for video playback to improve the learning experience.

**4. Discussion Forum: Let Students Discuss Exam-Related Topics**

**Purpose**:  
A discussion forum allows users to interact with each other, share ideas, ask questions, and collaborate on topics related to the exams. It also helps in building a community around the website.

**How to Implement**:

* **Forum Categories**: Create different discussion categories such as:
  + **General Questions**: Students can ask any questions related to the exam.
  + **Subject-Specific Discussions**: Topics on Pharmacology, Pharmaceutical Chemistry, etc.
  + **Exam Tips & Strategies**: Discussion on preparation strategies and how to tackle exam stress.
* **Posts and Replies**: Allow users to post new discussion threads and reply to others.
  + Ensure posts are moderated and adhere to community guidelines.
* **Upvoting/Downvoting**: Let users upvote or downvote posts based on relevance and usefulness.
* **Notification System**: Users can be notified when someone replies to their post or mentions them in a discussion.

**5. Admin Panel: To Manage Content, Users, and Monitor Site Activities**

**Purpose**:  
The admin panel is crucial for website maintenance and allows administrators to manage user accounts, monitor activities, and keep the content up to date.

**How to Implement**:

* **Content Management**: Admins can add, edit, and delete courses, study materials, and mock tests. They can also update video lessons.
  + Build a simple interface where admins can upload files, add new courses, and update existing content.
* **User Management**: Admins can view all user accounts, their progress, and even reset passwords if needed.
  + Add functionality for banning or suspending users who violate terms of service.
* **Activity Monitoring**: Admins can track user activity such as:
  + Number of logins.
  + Mock test attempts and results.
  + Discussions posted in the forum.

**6. Notifications: To Alert Users About Updates or Exam Tips**

**Purpose**:  
Notifications help engage users and keep them informed about new updates, exam tips, upcoming mock tests, or course updates. It can be a great way to remind students of important events.

**How to Implement**:

* **Push Notifications**: You can implement push notifications using tools like **Firebase Cloud Messaging (FCM)**.
  + When an important update (e.g., new study material) is posted, users can receive an instant notification.
* **Email Notifications**: Send email alerts to users when new mock tests, video lessons, or course updates are available.
  + You can use email services like **SendGrid** or **Amazon SES** to send bulk emails.
* **In-App Notifications**: Display notifications within the app itself (such as in the notification bell section) for alerts related to their account or new updates.

**Summary of Features**

| **Feature** | **Description** | **Implementation Ideas** |
| --- | --- | --- |
| **User Authentication** | Allows users to register, log in, and manage profiles securely. | Use JWT authentication for session management, bcrypt for password hashing. |
| **Courses & Study Materials** | Provides access to study resources, practice questions, and mock tests for exam preparation. | Organize materials into categories (subjects, topics). Store them in MongoDB, and allow browsing. |
| **Video Lessons** | Integrates video tutorials to explain exam topics and offer strategies. | Host on YouTube, Vimeo, or self-hosted via AWS S3 or Cloudinary. Provide interactive features for playback. |
| **Discussion Forum** | A place for users to ask questions, discuss topics, and collaborate. | Allow users to post threads, reply to discussions, and upvote helpful responses. Implement categories. |
| **Admin Panel** | Admins can manage content, users, and monitor site activities. | Admin interface to add/update/delete content, manage users, monitor activity. |
| **Notifications** | Alerts users about updates, exam tips, and important events. | Implement push notifications, email alerts, and in-app notifications using services like FCM or SendGrid. |

### ****API Structure Summary****

| **Feature** | **Endpoint** | **HTTP Method** | **Description** |
| --- | --- | --- | --- |
| **User Registration** | /api/user/register | POST | Register a new user |
| **User Login** | /api/user/login | POST | User login, returns JWT token |
| **Get User Profile** | /api/user/profile | GET | Get authenticated user's profile |
| **Update User Profile** | /api/user/profile | PUT | Update user's profile |
| **Create Course** | /api/courses | POST | Create a new course |
| **Get All Courses** | /api/courses | GET | Get all courses available |
| **Get Course by ID** | /api/courses/:courseId | GET | Get detailed info of a specific course |
| **Upload Study Material** | /api/materials | POST | Upload a study material |
| **Get Materials for Course** | /api/materials/:courseId | GET | Get materials related to a course |
| **Create Forum Post** | /api/forum | POST | Create a new discussion post |
| **Get All Forum Posts** | /api/forum | GET | Get all forum posts |
| **Reply to Forum Post** | /api/forum/:postId/reply | POST | Reply to a specific forum post |
| **Get Notifications** | /api/notifications | GET | Get all notifications for the authenticated user |
| **Send Notification** | /api/notifications | POST | Admin sends notifications to users |

### API ENDPOINTS for user ****Real-World Example of How Data is Stored****:

Let's look at a few example documents for different roles in the **User collection**.

#### **Example 1: A Student**

json

Copy code

{

"\_id": "unique\_id\_1",

"email": "student@example.com",

"password": "hashedpassword123",

"first\_name": "John",

"last\_name": "Doe",

"role": "student",

"status": "active",

"created\_at": "2025-01-01T12:00:00Z",

"updated\_at": "2025-01-01T12:00:00Z",

"profile\_picture": "https://example.com/images/johndoe.jpg",

"student\_id": "GPAT2025-12345",

"subscription\_plan": "Premium",

"subscription\_status": "Active"

}

#### **Example 2: A Teacher**

json

Copy code

{

"\_id": "unique\_id\_2",

"email": "teacher@example.com",

"password": "hashedpassword456",

"first\_name": "Jane",

"last\_name": "Smith",

"role": "teacher",

"status": "active",

"created\_at": "2024-12-15T09:00:00Z",

"updated\_at": "2024-12-15T09:00:00Z",

"profile\_picture": "https://example.com/images/janesmith.jpg",

"subscription\_plan": "Premium",

"subscription\_status": "Active"

}

#### **Example 3: An Admin**

json

Copy code

{

"\_id": "unique\_id\_3",

"email": "admin@example.com",

"password": "hashedadminpassword",

"first\_name": "Admin",

"last\_name": "User",

"role": "admin",

"status": "active",

"created\_at": "2023-11-01T14:00:00Z",

"updated\_at": "2023-11-01T14:00:00Z",

"profile\_picture": "https://example.com/images/adminuser.jpg"

}

#### **Example 4: A Guest**

json

Copy code

{

"\_id": "unique\_id\_4",

"email": "guest@example.com",

"password": null,

"first\_name": "Guest",

"last\_name": "User",

"role": "guest",

"status": "active",

"created\_at": "2025-01-01T15:00:00Z",

"updated\_at": "2025-01-01T15:00:00Z",

"profile\_picture": null

}

Aggregrate pipelones

{

$lookup: {

from: 'subjects', // Name of the 'subject' collection in MongoDB

localField: 'subject', // Field from Course model that refers to subject

foreignField: '\_id', // Field from Subject model that should match

as: 'subject' // The field name to store the resulting subject

}

},

The $lookup stage in the aggregation pipeline is used to **join** documents from two collections.

**from** specifies the collection you want to join ('subjects' in this case).

**localField** specifies the field in the Course document that holds the reference (foreign key) to another collection ('subject').

**foreignField** specifies the field in the Subject collection that should match the localField (i.e., \_id in Subject).

**as** is the name of the field in the result that will hold the joined documents (here we store it as subject).

#### Example:

Imagine you have a Course document like this:

{

"\_id": "1",

"title": "Intro to MongoDB",

"subject": "abc123" // This is a reference to a Subject document

}

And a Subject document like this:

{

"\_id": "abc123",

"name": "Database Management"

}

The $lookup operation would join them, resulting in the following document for the Course:

{

"\_id": "1",

"title": "Intro to MongoDB",

"subject": [

{

"\_id": "abc123",

"name": "Database Management"

}

]

}

$unwind: Flatten the Array

{

$unwind: {

path: '$subject', // Unwind subject array to work with a single object

preserveNullAndEmptyArrays: true // In case no subject is found

}

},

#### What is $unwind?

* The $unwind stage is used to **flatten** an array field in a document. After the $lookup operation, the subject field is an array (because it's a join), even if it only contains one item.
* $unwind will **break up** the array and return a document for each item inside it.

In the example above, the subject is returned as an array. $unwind will take the first object from the array and replace the array with that object.

If there’s no subject found (preserveNullAndEmptyArrays: true), the course document will still be included, but with null in place of subject.

#### After $unwind:

Your Course document will look like this:

{

"\_id": "1",

"title": "Intro to MongoDB",

"subject": {

"\_id": "abc123",

"name": "Database Management"

}

}

$project: Select the Final Fields

{

$project: { // Select only the fields that you want to return

title: 1,

description: 1,

duration: 1,

thumbnail: 1,

'subject.name': 1, // Subject name

'instructors.firstName': 1, // Instructor first name

'instructors.lastName': 1, // Instructor last name

ratings: 1, // Include ratings and reviews if needed

pricing: 1, // Include pricing if needed

topics: 1, // Include topics if needed

materials: 1, // Include materials if needed

mockTests: 1 // Include mock tests if needed

}

}

#### What is $project?

The $project stage allows us to **select** or **rename** fields from the documents. You can think of it like a **filter** to determine which fields should be included in the output.

* **1 means include** this field in the result.
* **0 means exclude** this field from the result.

Here, we are selecting the following fields:

* title, description, duration, thumbnail, etc., from the Course.
* The name from the subject (which was populated by $lookup).
* The firstName and lastName from the instructors.

If you don't include 1 for certain fields (like ratings, pricing, etc.), they will not appear in the result.

Final   
{

"\_id": "1",

"title": "Intro to MongoDB",

"description": "A course about MongoDB.",

"duration": "4 weeks",

"thumbnail": "thumbnail.jpg",

"subject": {

"name": "Database Management"

},

"instructors": {

"firstName": "John",

"lastName": "Doe"

},

"ratings": {

"average": 4.5,

"reviews": [

{

"user": "abc123",

"comment": "Great course!",

"rating": 5

}

]

},

"pricing": {

"originalPrice": 100,

"discounted