GrowthX API Documentation

Introduction

Welcome to the GrowthX API service. This API provides various endpoints for user and admin management, as well as assignment functionalities.

Installation and Setup

To get started with the GrowthX API Service, follow these steps:

Prerequisites

- **Node.js**: Ensure you have Node.js installed on your machine. You can download it from nodejs.org.
- **MongoDB**: Set up a MongoDB database. You can use a local instance or a cloud-based service like MongoDB Atlas.

Steps

1. **Clone the Repository** :
   git clone https://github.com/Hammad-4846/Student-Portal.git
2. **Install Dependencies**: Run the following command to install the necessary packages: npm install
3. **Environment Variables**: Create a .env file in the Config of the project and set the following environment variables:

   ```
   MONGO_URI
   PORT=4000
   JWT_EXPIRATION_TIME
   JWT_SECRET_KEY
   COOKIE_EXPIRE
   ```

4, **Run the Server**: Start the server with the command:

   npm start or node index.js

Base URL

All endpoints start with: /api/v1

Routes

User Routes

/auth/user/register
- Method: POST
- Description: Register a new user.

/auth/user/login
- Method: POST
- Description: Log in an existing user.

Admin Routes

/auth/admin/register
- Method: POST
- Description: Register a new admin.

/auth/admin/login
- Method: POST
- Description: Log in an existing admin.

Assignment Routes

/assignments
- Method: GET
- Description: Get all assignments for a specific admin (authenticated admin only).

/assignments/:id/accept
- Method: PUT
- Description: Accept an assignment (admin only).

/assignments/:id/reject
- Method: PUT
- Description: Reject an assignment (admin only).

/user/upload
- Method: POST
- Description: Upload an assignment (authenticated user only).

/user/alladmins
- Method: GET
- Description: Fetch all admins (only authorized users).

Middleware Information

1. isAuthenticatedUser: Checks if the user is authenticated.

2. checkRole: Checks if the user is an admin or user.

Authentication Method

Cookie-based authentication is used, as it automatically sends the token in the cookie.

Schema Information

1. User Schema: Solely for User and some methods.

2. Admin Schema: Solely for Admin and some methods.

The decision to create two schemas instead of one was made to avoid issues with duplicate names and emails when users and admins sign up through different portals.
also as the codebase grows, schemas can be modified differently if required.

but if we want to merge USER & ADMIN SCHEMA :-

```
const userSchema = new mongoose.Schema({

 username: {

  type: String,

  required: true,

  unique: true,

  minlength: 3,

 },


 email: {

  type: String,

  required: true,

  unique: true, // Ensures emails are unique among admins
  },

  password: {
    type: String,
    required: true,
    minlength: 6,
    select: false,
  },

//This will be added
role: {
 type: String,
 enum: ['user', 'admin'],
default: 'user',
},
  createdAt: {
    type: Date,
    default: Date.now,
  },
});
```