Project Documentation: Backend API for Frontend Integration

Overview

We will integrate your backend server with the front-end application. The backend must adhere to the API specifications outlined here.

The application consists of:

1. **Submission Form**:Users can submit details including name, email, age, post, description, and upload a file.
2. **Admin Login Page**: Allows admin authentication (email + password).
3. **Admin Dashboard:** Accessible after login and includes features to:
   - View a list of submissions with search, filters, sorting, and pagination.
   - Edit or remove submission items.

Features to Implement

-***Field Validation***:

- Name: First Name (FN) & Last Name (LN)
- Email
- Password
- Age
- Post: From a predefined constant list.
- Description: Text description.
- File Upload: Resume (PDF, DOCX only).

- Sorting: By creation date or post.

- Search: By post or name.

- Age Filters: `<` (less than) and `>` (greater than). For eg. Less than 30, More than 25, etc.

- Pagination: Server-side pagination.

- Authentication: JSON Web Token (JWT)-based login.

- Database: MongoDB to store application data.

- File Storage: S3 Bucket integration for file uploads.

API Specifications

Below is the structure of the API endpoints. Note that data types must be defined and validated by interviewers while implementing the backend.

Authentication

POST `/api/login` **(Admin Login)**

- *Description*: Authenticates the user and returns a JWT.
- *Request Body*:

```
{
  "email": "",
  "password": ""
}
```

- *Response*:

```
{
  "token": "",
  "user": {
    "id": "",
    "email": "",
     }
}
```

User Submission

USER API

POST `/api/submissions`

- *Description*: Creates a new submission.
- *Request Body*: {
  "name": {
   "firstName": "",
   "lastName": ""
  },
  "email": "",
  "password": "",
  "age": ,
  "post": "",
  "description": "",
  "file": "<File Upload>"
 }

- *Response*:

 {
  "message": "Submission created successfully",
  "submissionId": ""
}

ADMIN DASHBOARD APIs

GET `/api/submissions`

- *Description*: Retrieves submissions based on filters, search, sorting, and pagination.
- *Query Parameters*:
  - `search`: Text search by name or post.
  - `post`: Filter by specific post.
  - `age[lt]`: Filter by age less than.
  - `age[gt]`: Filter by age greater than.
  - `sort`: Sorting by `creation_date` or `post`.

  - `skip`: Page number (pagination).
  - `limit`: Number of items per page.

- *Response*:
  {
    "total": ,
    "skip": ,
    "limit": ,
    "submissions": [
      {
        "id": "",
        "name": {
          "firstName": "",
          "lastName": ""
        },
        "email": "",
        "age": ,
        "post": "",
        "description": "",
        "fileUrl": "",
        "creationDate": ""
      }
    ]
  }

PATCH `/api/submissions/:id`

- *Description:* Edits a specific submission.
- *Request Body:*
  {
    "name": {
      "firstName": "",
      "lastName": ""
    },
    "email": "",
    "age": ,
    "post": "",
    "description": "",
    "file": "<File Upload>"
  }

- Response:
  ```
  {
    "message": "Submission updated successfully"
  }
  ```

DELETE `/api/submissions/:id`

- *Description*: Removes a specific submission.
- *Response*:
  ```
  {
    "message": "Submission deleted successfully"
  }
  ```

Authentication Workflow

- Use JWT for authentication:
  1. On login, generate a token and send it to the client.
  2. Protect API routes using middleware to verify the token.

S3 Bucket Integration

- Use AWS S3 to upload and store files securely.
- Store the file URL in MongoDB for retrieval.
- Ensure appropriate file validation for type (PDF, DOCX).

Notes

- API responses must be well-structured, including appropriate status codes (`200`, `201`, `400`, `401`, `404`, `500`).