roone858 / **School-Managment-System** Private

Code Issues Pull requests Actions Projects Security Insights Se

main School-Managment-System / api / Go to file Add file ...

roone858 Edit readme File `c01fedc` 3 hours ago **History**

..

src	edit readme file	12 hours ago
README.md	Edit readme File	3 hours ago
package-lock.json	add grade model and controller	2 days ago
package.json	add grade model and controller	2 days ago
sql.sql	add grade model and controller	2 days ago

Overview 🔷

A School Management System API is a type of application programming interface (API) that is specifically designed for use with school management systems. Its purpose is to provide programmatic access to the features and data of the school management system, enabling developers to create custom applications that can interact with the system.

API Features 🔥

A school management API have many features to help manage various aspects of a school. Here are some common features that a school management API include:

- Student Management - ability to create, update and delete student records, including
- personal information, contact details, enrollment status, and academic progress.
- Teacher Management - ability to create, update and delete teacher records, including
- personal information, contact details, qualifications, and courses taught.
- Course Management - ability to create, update and delete course records, including course descriptions, course schedules, and course materials.

- Attendance Tracking - ability to record and view attendance for each student in a class. These are just some of the features that a school management API can have. The specific features will depend on the requirements of the school or educational institution that is using the API.

Endpoints 💻

1. `/api/students` :
   - `GET` : get a list of all students
   - `POST` : create a new student
2. `/api/students/:id` :
   - `GET` : get a specific student by ID
   - `PUT` : update a specific student by ID
   - `DELETE` : delete a specific student by ID
3. `/api/teachers` :
   - `GET` : get a list of all teachers
   - `POST` : create a new teacher
4. `/api/teachers/:id` :
   - `GET` : get a specific teacher by ID
   - `PUT` : update a specific teacher by ID
   - `DELETE` : delete a specific teacher by ID
5. `/api/courses` :
   - `GET` : get a list of all courses
   - `POST` : create a new course
6. `/api/courses/:id` :
   - `GET` : get a specific course by ID
   - `PUT` : update a specific course by ID
   - `DELETE` : delete a specific course by ID
7. `/api/enrollments` :
   - `GET` : get a list of all enrollments
   - `POST` : create a new enrollment
8. `/api/enrollments/:id` :
   - `GET` : get a specific enrollment by ID
   - `PUT` : update a specific enrollment by ID
   - `DELETE` : delete a specific enrollment by ID
9. `/api/attendance` :
   - `GET` : get a list of all attendance records
   - `POST` : create a new attendance record

10. `/api/attendance/:id` :
    - `GET` : get a specific attendance record by ID
    - `PUT` : update a specific attendance record by ID
    - `DELETE` : delete a specific attendance record by ID
11. `/api/grades` :
    - `GET` : get a list of all grades
    - `POST` : create a new grade
12. `/api/grades/:id` :
    - `GET` : get a specific grade by ID
    - `PUT` : update a specific grade by ID
    - `DELETE` : delete a specific grade by ID

may also want to include authentication endpoints, such as:

- `POST /login` : Authenticate a user and generate an access token.
- `POST /logout` : Invalidate an access token and log the user out

Libraries 💡

The API is built with Node.js and Express, and uses a PostgreSQL database to store data. The following libraries and SDKs are used:

- Express - A fast, unopinionated, minimalist web framework for Node.js.
- Pg - A PostgreSQL client for Node.js. pg-pool: A connection pool for PostgreSQL clients.
- Nodemon - A tool that helps develop Node.js-based applications by automatically restarting the node application when file changes in the directory are detected.
- Bcrypt - A library for hashing passwords.
- Jsonwebtoken - A library for creating and verifying JSON Web Tokens (JWTs).
- body-parser - A middleware for handling HTTP request bodies.

README.md

To install the necessary libraries and SDKs, run the following command in the project directory:

```
npm install express pg pg-pool nodemon bcrypt jsonwebtoken body-parser cors
```

Project Structure [api] 📂

```
# api Package
.
├── src             # Our core frontend code consisting of all views and
react components
|   ├── models      # This directory contains modules that define the
schema for the data and interact with the database.
|   |    ├── student.model.js
|   |    ├── teacher.model.js
|   |    ├── course.model.js
|   |    ├── attendance.model.js
|   |    ├── parent.model.js
|   |    ├── grade.model.js
|   |    └── enrollment.model.js
|   ├── controllers           # This directory contains modules that
handle the logic of the API endpoints .
|   |    ├── parent.controller.js
|   |    ├── student.controller.js
|   |    ├── teacher.controller.js
|   |    ├── course.controller.js
|   |    ├── attendance.controller.js
|   |    ├── enrollment.controller.js
|   |    └── grade.controller.js
|   ├── middleware     # Contains UI component files for dialog boxes,
snackbars, single user in inbox etc
|   ├── routes         # This directory contains modules that define the
API routes and connect them to the controller methods.
|   ├── server.js      # This is the main entry point of the application,
where we initialize the Express app,  and set up middleware.
|   └── db.js          # connect to the database
├── .env              # This file contains sensitive information (such as
database credentials) and is not committed to version control.
├── package.json      # This file lists the project dependencies and
metadata.
└── package-lock.json
```

Usage on your Local 📝

1. Prepare env
    - add a `.ENV` file in the root directory and set the missing `###` environment parameters

```
POSTGRES_HOST=127.0.0.1
POSTGRES_DB=school
POSTGRES_USER=postgres
```

```
POSTGRES_PASSWORD=1234
POSTGRES_PORT=5432
TOKEN_SECRET=my secret
SALT_ROUNDS=12
```

2. Create Databases
    - connect to the default postgres database as the server's root user `psql -U postgres`
    - In psql run the following to create a user
    - `CREATE USER school_user WITH PASSWORD 'password123';`
    - In psql run the following to create the database
    - `CREATE DATABASE school;`
    - Connect to the databases and grant all privileges
    - `\c school`
3. Migrate Database
    - Navigate to the root directory and run the command below to migrate the database `npx db-migrate up`
4. Set up
    - `npm install` to install all dependencies
5. Start the app
    - `npm run dev` to start the app and get access via http://localhost:3000
6. Running Ports
    - After start up, the server will start on port `3000` and the database on port `5432`

Response Formats

- The API returns all data in JSON format. Responses will have a Content-Type header of application/json.
- Examples
    - GET /students

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
[
    {
        "id": 1,
        "firstname": "Mahmoud",
        "lastname": "Gamal",
        "email": "roone@gmail.com",
        "gender": "Male",
        "phone": "0115043454",
        "dateofbirth": "1998-12-31T22:00:00.000Z",
        "address": "Assiut - Egypt",
        "parentid": null
    },
    {
        "id": 3,
        "firstname": "Hesham",
        "lastname": "Abbas",
        "email": "habbas@gmail.com",
        "gender": "male",
        "phone": "324234333",
        "dateofbirth": "1999-12-31T22:00:00.000Z",
        "address": "Assiut - Egypt",
        "parentid": 1
    }
]
```

- POST /students

```
HTTP/1.1 201 Created
Content-Type: application/json

  {
        "id": 3,
        "firstname": "Hesham",
        "lastname": "Abbas",
        "email": "habbas@gmail.com",
        "gender": "male",
        "phone": "324234333",
        "dateofbirth": "1999-12-31T22:00:00.000Z",
        "address": "Assiut - Egypt",
        "parentid": 1
    }
```

In the above examples, the responses are shown in JSON format with the appropriate HTTP status code and content type headers. The data returned in the response depends on the endpoint being called and the parameters passed in the request.

Error Handling ✍️

If an error occurs while processing a request, the API will return an error response with an appropriate status code and error message. The following status codes may be returned:

- `400 Bad Request` : The request was invalid or could not be understood by the server.
- `401 Unauthorized` : The client is not authorized to access the requested resource.
- `403 Forbidden` : The client does not have permission to access the requested resource.
- `404 Not Found` : The requested resource could not be found.
- `500 Internal Server Error` : An unexpected error occurred on the server.

Give feedback