Certainly! The code provides endpoints for registering an admin, logging in an admin, and retrieving admin details in a Node.js application using MongoDB. Below is a detailed explanation of each part of the code:
Backend > controllers >

1.Admin-controller.js
### Required Modules
const bcrypt = require('bcrypt');
const Admin = require('../models/adminSchema.js');
const Sclass = require('../models/sclassSchema.js');
const Student = require('../models/studentSchema.js');
const Teacher = require('../models/teacherSchema.js');
const Subject = require('../models/subjectSchema.js');
const Notice = require('../models/noticeSchema.js');
const Complain = require('../models/complainSchema.js');
```
- `bcrypt`: A library used to hash passwords.
- `Admin`, `Sclass`, `Student`, `Teacher`, `Subject`, `Notice`, `Complain`: Mongoose models representing collections in the MongoDB database.

**### Admin Registration**
1. The `adminRegister` function handles the registration of a new admin.
2. It extracts `email`, `schoolName`, and `password` from the request body.
3. It checks if an admin with the same email or school name already exists in the database.
4. If either already exists, it sends a 400 status code with an appropriate message.
5. If not, it hashes the password using bcrypt and creates a new admin document with the provided data and hashed password.
6. It saves the new admin to the database, removes the password from the response, and sends a 201 status code with the admin details.
7. If any error occurs, it catches the error and sends a 500 status code with the error message.

**### Admin Login**

1. The `adminLogIn` function handles the login of an admin.
2. It extracts `email` and `password` from the request body.
3. It checks if both email and password are provided; if not, it sends a 400 status code with an error message.
4. It looks for an admin with the provided email in the database.
5. If the admin is not found, it sends a 404 status code with an error message.
6. It compares the provided password with the stored hashed password using bcrypt.
7. If the passwords do not match, it sends a 400 status code with an error message.
8. If the passwords match, it removes the password from the response and sends a 200 status code with the admin details.
9. If any error occurs, it catches the error and sends a 500 status code with the error message.

**### Get Admin Details**
1. The `getAdminDetail` function retrieves the details of an admin by their ID.
2. It extracts the admin ID from the request parameters.
3. It searches for the admin with the provided ID in the database.
4. If the admin is not found, it sends a 404 status code with an error message.
5. If the admin is found, it removes the password from the response and sends a 200 status code with the admin details.
6. If any error occurs, it catches the error and sends a 500 status code with the error message.

**### Exporting the Functions**
module.exports = { adminRegister, adminLogIn, getAdminDetail };
```
- Exports the three functions (`adminRegister`, `adminLogIn`, and `getAdminDetail`) so they can be used in other parts of the application (e.g., routing).

2.class-controller.js
Certainly! The provided code defines several functions for handling CRUD operations related to school classes (`Sclass`), students (`Student`), subjects (`Subject`), and teachers (`Teacher`) in a Node.js application using MongoDB. Here's an explanation of each function:
###Required Modules:
const Sclass = require('../models/sclassSchema.js');
const Student = require('../models/studentSchema.js');
const Subject = require('../models/subjectSchema.js');
const Teacher = require('../models/teacherSchema.js');
- These are Mongoose models representing the collections in the MongoDB database for classes, students, subjects, and teachers.

###Create a New class
1. Creates a new class with the provided `sclassName` and `adminID`.
2. Checks if a class with the same name already exists in the school.
3. If it exists, sends a message indicating the class name already exists.
4. If not, saves the new class and sends the result.
5. Catches and handles errors, sending a 500 status code if any occur.

###List All Classes for a School
1. Retrieves all classes for a specific school.
2. If classes are found, sends the list of classes.
3. If no classes are found, sends a message indicating no classes were found.
4. Catches and handles errors, sending a 500 status code if any occur.

###Get Class Details
1. Retrieves details of a class by its ID.
2. Populates the `school` field with the `schoolName`.

3. If the class is found, sends the class details.
4. If not, sends a message indicating no class was found.
5. Catches and handles errors, sending a 500 status code if any occur.

###Get Students in a Class

1. Retrieves all students in a specific class by its ID.
2. If students are found, removes their passwords from the response and sends the list.
3. If no students are found, sends a message indicating no students were found.
4. Catches and handles errors, sending a 500 status code if any occur.

###Delete a Class

1. Deletes a specific class by its ID.
2. If the class is not found, sends a message indicating the class was not found.
3. Deletes all students, subjects, and teachers associated with the class.
4. Sends the deleted class details.
5. Catches and handles errors, sending a 500 status code if any occur.

###Delete All Classes for a School

1. Deletes all classes for a specific school.
2. If no classes are found to delete, sends a message indicating no classes were found.
3. Deletes all students, subjects, and teachers associated with the school.
4. Sends the deleted classes details.
5. Catches and handles errors, sending a 500 status code if any occur.

###Exporting the Functions

```
module.exports = { sclassCreate, sclassList, deleteSclass,
deleteSclasses, getSclassDetail, getSclassStudents };
```

- Exports the functions so they can be used in other parts of the application (e.g., routing).

3.complain-controller.js

The provided code defines two functions for handling complaints (`complainCreate` and `complainList`) in a Node.js application using MongoDB. Here's a detailed explanation of each function:

Required Modules

```
const Complain = require('../models/complainSchema.js');
```

- Imports the `Complain` Mongoose model, representing the collection of complaints in the MongoDB database.

Create a New Complaint

1. The `complainCreate` function handles the creation of a new complaint.
2. It creates a new `Complain` document using the request body (`req.body`), which should contain the complaint details.
3. It saves the new complaint to the database using `complain.save()`.
4. If the save operation is successful, it sends the saved complaint as a response.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

List All Complaints for a School

1. The `complainList` function retrieves all complaints for a specific school.
2. It uses `Complain.find({ school: req.params.id })` to find complaints associated with the school ID provided in the request parameters (`req.params.id`).
3. The `populate("user", "name")` method is used to populate the `user` field in the complaints with the user's name.
4. If complaints are found, it sends the list of complaints as a response.
5. If no complaints are found, it sends a message indicating that no complaints were found.
6. If an error occurs, it catches the error and sends a 500 status code with the error message.

Exporting the Functions

```
module.exports = { complainCreate, complainList };
```

- Exports the `complainCreate` and `complainList` functions so they can be used in other parts of the application, such as routing.

Summary

- `complainCreate`: This function creates a new complaint and saves it to the database. It sends the saved complaint or an error message as a response.
- `complainList`: This function retrieves all complaints for a specific school, populates the user field with user names, and sends the list of complaints or an error message as a response.

4.notice-controller.js

The provided code defines several functions for handling notices (`noticeCreate`, `noticeList`, `updateNotice`, `deleteNotice`, `deleteNotices`) in a Node.js application using MongoDB. Here is a detailed explanation of each function:

Required Modules

```
const Notice = require('../models/noticeSchema.js');
```

- Imports the `Notice` Mongoose model, representing the collection of notices in the MongoDB database.

Create a New Notice

1. The `noticeCreate` function handles the creation of a new notice.
2. It creates a new `Notice` document using the request body (`req.body`), including the `adminID` as the `school` field.
3. It saves the new notice to the database using `notice.save()`.
4. If the save operation is successful, it sends the saved notice as a response.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

List All Notices for a School

1. The `noticeList` function retrieves all notices for a specific school.
2. It uses `Notice.find({ school: req.params.id })` to find notices associated with the school ID provided in the request parameters (`req.params.id`).
3. If notices are found, it sends the list of notices as a response.
4. If no notices are found, it sends a message indicating that no notices were found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Update a Notice

1. The `updateNotice` function handles updating a notice by its ID.
2. It uses `Notice.findByIdAndUpdate` to find the notice by ID (`req.params.id`) and updates it with the new data provided in the request body (`req.body`).
3. The `{ new: true }` option ensures that the updated document is returned.
4. If the update operation is successful, it sends the updated notice as a response.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete a Notice

1. The `deleteNotice` function handles deleting a notice by its ID.
2. It uses `Notice.findByIdAndDelete` to find and delete the notice by ID (`req.params.id`).
3. If the delete operation is successful, it sends the deleted notice as a response.
4. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete All Notices for a School

1. The `deleteNotices` function handles deleting all notices for a specific school.
2. It uses `Notice.deleteMany({ school: req.params.id })` to find and delete all notices associated with the school ID provided in the request parameters (`req.params.id`).
3. If no notices are found to delete, it sends a message indicating that no notices were found.
4. If the delete operation is successful, it sends the result of the deletion.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Exporting the Functions

```
  module.exports = { noticeCreate, noticeList, updateNotice,
deleteNotice, deleteNotices };
```

- Exports the functions so they can be used in other parts of the application (e.g., routing).

Summary

- `noticeCreate`: Creates a new notice and saves it to the database.
- `noticeList`: Retrieves all notices for a specific school.
- `updateNotice`: Updates a specific notice by its ID.
- `deleteNotice`: Deletes a specific notice by its ID.
- `deleteNotices`: Deletes all notices for a specific school.

5.Student-controller.js

The provided code defines several functions for handling student-related operations (`studentRegister`, `studentLogIn`, `getStudents`, `getStudentDetail`, `deleteStudent`, `deleteStudents`, `deleteStudentsByClass`, `updateStudent`, `updateExamResult`, `studentAttendance`, `clearAllStudentsAttendanceBySubject`, `clearAllStudentsAttendance`, `removeStudentAttendanceBySubject`, `removeStudentAttendance`) in a Node.js application using MongoDB. Here's a detailed explanation of each function:

Required Modules

- Imports the `bcrypt` library for password hashing.
- Imports the `Student` and `Subject` Mongoose models, representing the collections of students and subjects in the MongoDB database.

Register a New Student

1. The `studentRegister` function handles the registration of a new student.
2. It generates a salt and hashes the student's password using `bcrypt`.
3. It checks if a student with the same roll number already exists in the specified class and school.
4. If the student exists, it sends a message indicating that the roll number already exists.
5. If the student does not exist, it creates a new `Student` document using the request body and the hashed password, then saves it to the database.
6. It sends the saved student (excluding the password) as a response or catches any errors and sends a 500 status code with the error message.

Student Login

1. The `studentLogIn` function handles student login.
2. It finds the student by roll number and name.
3. If the student exists, it compares the provided password with the stored hashed password using `bcrypt`.
4. If the passwords match, it populates the student's school and class details, removes sensitive fields, and sends the student as a response.
5. If the passwords do not match or the student is not found, it sends an appropriate message.
6. If an error occurs, it catches the error and sends a 500 status code with the error message.

Get All Students for a School

1. The `getStudents` function retrieves all students for a specific school.
2. It finds students by the school ID provided in the request parameters and populates their class details.
3. If students are found, it removes the password field from each student and sends the list of students as a response.
4. If no students are found, it sends a message indicating that no students were found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Get Student Detail

The `getStudentDetail` function retrieves the details of a specific student by their ID.

1. It finds the student by ID and populates their school, class, exam results, and attendance details.
2. If the student is found, it removes the password field and sends the student as a response.
3. If no student is found, it sends a message indicating that no student was found.
4. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete a Student

1. The `deleteStudent` function deletes a specific student by their ID.
2. It finds and deletes the student by ID.
3. If the delete operation is successful, it sends the deleted student as a response.
4. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete All Students for a School

1. The `deleteStudents` function deletes all students for a specific school.
2. It finds and deletes students by the school ID provided in the request parameters.
3. If no students are found to delete, it sends a message indicating that no students were found.
4. If the delete operation is successful, it sends the result of the deletion.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete All Students for a Class

1. The `deleteStudentsByClass` function deletes all students for a specific class.
2. It finds and deletes students by the class ID provided in the request parameters.
3. If no students are found to delete, it sends a message indicating that no students were found.
4. If the delete operation is successful, it sends the result of the deletion.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Update a Student

1. The `updateStudent` function updates the details of a specific student by their ID.
2. If the request body contains a password, it hashes the new password.

3. It updates the student with the new data provided in the request body and returns the updated student.
4. If the update operation is successful, it removes the password field and sends the updated student as a response.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Update Exam Results for a Student

The `updateExamResult` function updates the exam results for a specific student by their ID.

1. It finds the student by ID and checks if the student exists.
2. If the student exists, it checks if the exam result for the specified subject already exists.
3. If the exam result exists, it updates the marks obtained.
4. If the exam result does not exist, it adds a new exam result for the subject.
5. It saves the updated student and sends the result as a response or catches any errors and sends a 500 status code with the error message.

Manage Student Attendance

1. The `studentAttendance` function manages the attendance of a specific student by their ID.
2. It finds the student by ID and checks if the student exists.
3. It finds the subject by ID.
4. It checks if attendance for the specified subject and date already exists.
5. If the attendance exists, it updates the attendance status.
6. If the attendance does not exist, it checks if the student has attended the maximum number of sessions for the subject.
7. If the maximum attendance limit is not reached, it adds a new attendance record.
8. It saves the updated student and sends the result as a response or catches any errors and sends a 500 status code with the error message.

Clear Attendance for All Students by Subject

1. The `clearAllStudentsAttendanceBySubject` function clears the attendance for all students by subject ID.
2. It finds and removes attendance records for the specified subject from all students.
3. If the operation is successful, it sends the result as a response or catches any errors and sends a 500 status code with the error message.

Clear All Attendance for All Students

1. The `clearAllStudentsAttendance` function clears all attendance records for all students in a specific school.
2. It finds and removes all attendance records for students in the specified school.
3. If the operation is successful, it sends the result as a response or catches any errors and sends a 500 status code with the error message.

Remove Attendance for a Specific Student by Subject

1. The `removeStudentAttendanceBySubject` function removes attendance records for a specific student by subject ID.
2. It finds and removes attendance records for the specified subject from the specified student.
3. If the operation is successful, it sends the result as a response or catches any errors and sends a 500 status code with the error message.

Remove All Attendance for a Specific Student

1. The `removeStudentAttendance` function removes all attendance records for a specific student.
2. It finds and removes all attendance records from the specified student.
3. If the operation is successful, it sends the result as a response or catches any errors and sends a 500 status code with the error message.

Exporting the Functions

- Exports the functions so they can be used in other parts of the application (e.g., routing).

Summary

- `studentRegister`: Registers a new student and saves it to the database.
- `studentLogIn`: Handles student login and authentication.
- `getStudents`: Retrieves all students for a specific school.
- `getStudentDetail`: Retrieves the details of a specific student by their ID.
- `deleteStudent`: Deletes a specific student by their ID.
- `deleteStudents`: Deletes all students for a specific school.
- `deleteStudentsByClass`: Deletes all students for a specific class.
- `updateStudent`: Updates the details of a specific student by their ID.
- `updateExamResult`: Updates the exam results for a specific student by their ID.
- `studentAttendance`: Manages the attendance of a specific student by their ID.
- `clearAllStudentsAttendanceBySubject`: Clears attendance for all students by subject ID.
- `clearAllStudentsAttendance`: Clears all attendance records for all students in a specific school.
- `removeStudentAttendanceBySubject`: Removes attendance records for a specific student by subject ID.
- `removeStudentAttendance`: Removes all attendance records for a specific student.

6.subject-controller.js

The provided code defines several functions for handling subject-related operations (`subjectCreate`, `allSubjects`, `classSubjects`, `freeSubjectList`, `getSubjectDetail`, `deleteSubject`, `deleteSubjects`, `deleteSubjectsByClass`) in a Node.js application using MongoDB. Here's a detailed explanation of each function:

Required Modules

- Imports the `Subject`, `Teacher`, and `Student` Mongoose models, representing the collections of subjects, teachers, and students in the MongoDB database.

Create Subjects

1. The `subjectCreate` function handles the creation of new subjects.
2. It maps the `subjects` array from the request body to extract the subject details.
3. It checks if a subject with the same `subCode` already exists in the specified school.
4. If the subject exists, it sends a message indicating that the `subCode` must be unique.
5. If the subject does not exist, it creates new subject documents and saves them to the database.
6. It sends the saved subjects as a response or catches any errors and sends a 500 status code with the error message.

Get All Subjects for a School

1. The `allSubjects` function retrieves all subjects for a specific school.
2. It finds subjects by the school ID provided in the request parameters and populates their class details.
3. If subjects are found, it sends the list of subjects as a response.
4. If no subjects are found, it sends a message indicating that no subjects were found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Get Subjects for a Class

1. The `classSubjects` function retrieves all subjects for a specific class.
2. It finds subjects by the class ID provided in the request parameters.
3. If subjects are found, it sends the list of subjects as a response.
4. If no subjects are found, it sends a message indicating that no subjects were found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Get Free Subjects (Subjects Without Assigned Teachers)

1. The `freeSubjectList` function retrieves subjects for a specific class that do not have assigned teachers.
2. It finds subjects by the class ID provided in the request parameters and checks if the `teacher` field does not exist.
3. If subjects are found, it sends the list of subjects as a response.
4. If no subjects are found, it sends a message indicating that no subjects were found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Get Subject Details

1. The `getSubjectDetail` function retrieves the details of a specific subject by its ID.
2. It finds the subject by ID and checks if the subject exists.
3. If the subject exists, it populates the class and teacher details and sends the subject as a response.
4. If no subject is found, it sends a message indicating that no subject was found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete a Subject

1. The `deleteSubject` function deletes a specific subject by its ID.
2. It finds and deletes the subject by ID.
3. It updates the teachers to unset the `teachSubject` field if they were teaching the deleted subject.
4. It removes the deleted subject from the `examResult` and `attendance` arrays of all students.
5. If the delete operation is successful, it sends the deleted subject as a response.
6. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete All Subjects for a School

1. The `deleteSubjects` function deletes all subjects for a specific school.
2. It finds and deletes subjects by the school ID provided in the request parameters.
3. It updates the teachers to unset the `teachSubject` field if they were teaching the deleted subjects.
4. It sets the `examResult` and `attendance` fields to null in all students.
5. If the delete operation is successful, it sends the deleted subjects as a response.
6. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete All Subjects for a Class

1. The `deleteSubjectsByClass` function deletes all subjects for a specific class.
2. It finds and deletes subjects by the class ID provided in the request parameters.
3. It updates the teachers to unset the `teachSubject` field if they were teaching the deleted subjects.

4. It sets the `examResult` and `attendance` fields to null in all students.
5. If the delete operation is successful, it sends the deleted subjects as a response.
6. If an error occurs, it catches the error and sends a 500 status code with the error message.

Exporting the Functions

```
module.exports = { subjectCreate, freeSubjectList, classSubjects,
getSubjectDetail, deleteSubjectsByClass, deleteSubjects,
deleteSubject, allSubjects };
```

- Exports the functions so they can be used in other parts of the application (e.g., routing).

Summary

- **`subjectCreate`**: Creates new subjects and ensures `subCode` uniqueness within the school.
- **`allSubjects`**: Retrieves all subjects for a specific school.
- **`classSubjects`**: Retrieves all subjects for a specific class.
- **`freeSubjectList`**: Retrieves subjects for a specific class that do not have assigned teachers.
- **`getSubjectDetail`**: Retrieves the details of a specific subject by its ID.
- **`deleteSubject`**: Deletes a specific subject by its ID and updates related teachers and students.
- **`deleteSubjects`**: Deletes all subjects for a specific school and updates related teachers and students.
- **`deleteSubjectsByClass`**: Deletes all subjects for a specific class and updates related teachers and students.

7.teacher-controller.js

The provided code defines several functions for managing teacher-related operations (`teacherRegister`, `teacherLogIn`, `getTeachers`, `getTeacherDetail`, `updateTeacherSubject`, `deleteTeacher`, `deleteTeachers`, `deleteTeachersByClass`, `teacherAttendance`) in a Node.js application using MongoDB. Here's a detailed explanation of each function:

Required Modules

- Imports the `bcrypt` library for hashing passwords.
- Imports the `Teacher` and `Subject` Mongoose models, representing the collections of teachers and subjects in the MongoDB database.

Register a Teacher

1. The `teacherRegister` function handles the registration of a new teacher.
2. It hashes the password using `bcrypt` and creates a new `Teacher` document with the provided details.
3. It checks if a teacher with the same email already exists.
4. If the teacher exists, it sends a message indicating that the email already exists.
5. If the teacher does not exist, it saves the new teacher document to the database.
6. It updates the `teachSubject` in the `Subject` collection to associate it with the newly registered teacher.
7. It removes the password from the response and sends the saved teacher as a response.
8. If an error occurs, it catches the error and sends a 500 status code with the error message.

Teacher Login

1. The `teacherLogIn` function handles the login process for a teacher.
2. It finds a teacher by the email provided in the request body.
3. If the teacher is found, it compares the provided password with the stored hashed password using `bcrypt`.
4. If the password is valid, it populates the related `teachSubject`, `school`, and `teachSclass` fields and removes the password from the response.
5. If the password is invalid, it sends a message indicating that the password is invalid.
6. If the teacher is not found, it sends a message indicating that the teacher is not found.
7. If an error occurs, it catches the error and sends a 500 status code with the error message.

Get All Teachers for a School

1. The `getTeachers` function retrieves all teachers for a specific school.
2. It finds teachers by the school ID provided in the request parameters and populates their `teachSubject` and `teachSclass` details.
3. If teachers are found, it removes the password from each teacher and sends the list of teachers as a response.
4. If no teachers are found, it sends a message indicating that no teachers were found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Get Teacher Details

1. The `getTeacherDetail` function retrieves the details of a specific teacher by their ID.
2. It finds the teacher by ID and populates the related `teachSubject`, `school`, and `teachSclass` fields.
3. If the teacher is found, it removes the password and sends the teacher details as a response.
4. If no teacher is found, it sends a message indicating that no teacher was found.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Update Teacher's Subject

1. The `updateTeacherSubject` function updates the subject taught by a specific teacher.
2. It finds the teacher by ID and updates the `teachSubject` field with the provided subject ID.
3. It also updates the `teacher` field in the `Subject` collection to associate it with the updated teacher.
4. It sends the updated teacher as a response.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete a Teacher

1. The `deleteTeacher` function deletes a specific teacher by their ID.
2. It finds and deletes the teacher by ID.
3. It updates the `teacher` field in the `Subject` collection to unset it if the teacher was teaching the subject.
4. If the delete operation is successful, it sends the deleted teacher as a response.
5. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete All Teachers for a School

1. The `deleteTeachers` function deletes all teachers for a specific school.
2. It finds and deletes teachers by the school ID provided in the request parameters.
3. If no teachers are found to delete, it sends a message indicating this.
4. It updates the `teacher` field in the `Subject` collection to unset it if the teachers were teaching the subjects.
5. If the delete operation is successful, it sends the deletion result as a response.
6. If an error occurs, it catches the error and sends a 500 status code with the error message.

Delete All Teachers for a Class

1. The `deleteTeachersByClass` function deletes all teachers for a specific class.
2. It finds and deletes teachers by the class ID provided in the request parameters.
3. If no teachers are found to delete, it sends a message indicating this.
4. It updates the `teacher` field in the `Subject` collection to unset it if the teachers were teaching the subjects.
5. If the delete operation is successful, it sends the deletion result as a response.
6. If an error occurs, it catches the error and sends a 500 status code with the error message.

Record Teacher Attendance

1. The `teacherAttendance` function records the attendance status for a teacher on a specific date.
2. It finds the teacher by ID.
3. If the teacher is not found, it sends a message indicating this.
4. It checks if attendance for the given date already exists.
5. If attendance exists, it updates the status; otherwise, it adds a new attendance record.
6. It saves the updated teacher document and sends the result as a response.
7. If an error occurs, it catches the error and sends a 500 status code with the error message.

Exporting the Functions

- Exports the functions so they can be used in other parts of the application, such as in routing.

Summary

- **`teacherRegister`**: Registers a new teacher and assigns them to a subject.
- **`teacherLogIn`**: Logs in a teacher by validating their email and password.
- **`getTeachers`**: Retrieves all teachers for a specific school.
- **`getTeacherDetail`**: Retrieves details of a specific teacher by their ID.
- **`updateTeacherSubject`**: Updates the subject taught by a specific teacher.
- **`deleteTeacher`**: Deletes a specific teacher and updates related subjects.
- **`deleteTeachers`**: Deletes all teachers for a specific school and updates related subjects.
- **`deleteTeachersByClass`**: Deletes all teachers for a specific class and updates related subjects.
- **`teacherAttendance`**: Records the attendance status for a teacher on a specific date.