**Question**

**Tables**

Student Table

| ID\* | Name | Age | Grade | Class ID |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Teacher Table

| ID\* | Name | Subject |
| --- | --- | --- |
|  |  |  |

Classes Table

| ID\* | Subject | TeacherID | No of Students |
| --- | --- | --- | --- |
|  |  |  |  |

**End Points:-**

* /totalStudents
* /avgAgeOfStudents
* /noOfClasses
* POST /students - Add a new student.
* GET /students/{id} - Retrieve student details.
* GET /students?cursor={cursor} - List students using cursor-based pagination for efficient data retrieval.
* PUT /students/{id} - Update student information.
* DELETE /students/{id} - Remove a student.

**Task**: Develop a real-time chat system using Socket.IO with Node.js

* Set up a Node.js server with Socket.IO.
* Implement client-side scripts to establish a WebSocket connection to the server.
* Ensure the server handles incoming messages and broadcasts them to all connected clients.
* Test the functionality by setting up multiple clients (could be simple Node.js scripts) to send and receive messages.

**Solution**

**Approach**: My approach involved creating three distinct parts of a full-stack Node.js project.

* Data Aggregation System: Utilized Node.js and Express to create an API for aggregating data from a MySQL database.
* Real-Time Communication Feature: Implemented a real-time chat system using Socket.IO for bidirectional communication between clients and the server.
* Basic API with Scalability Considerations: Developed a RESTful API for user data operations with scalability in mind, incorporating cursor-based pagination.

**Challenges**

* Data Aggregation System: Faced challenges in designing efficient database queries for data aggregation.
* Real-Time Communication Feature: Overcame difficulties in managing and broadcasting real-time messages among connected clients.
* Basic API: Tackled issues related to pagination and ensuring scalable data retrieval.

**Solutions**

* Data Aggregation System: Optimized SQL queries for better performance and utilized asynchronous programming for non-blocking operations.
* Real-Time Communication Feature: Implemented Socket.IO for real-time communication and handled events for message broadcasting.
* Basic API: Incorporated cursor-based pagination to efficiently handle large data sets.

**Conclusion**

This full-stack Node.js project demonstrates proficiency in building various components, including data aggregation, real-time communication, and scalable APIs. The challenges encountered were addressed with thoughtful solutions, contributing to the overall success of the project.