Dear Students,

Please refer to the attached document as a sample set of interview-oriented questions based on Node.js, Express.js, MongoDB, and REST APIs.

These questions are designed to help you understand and practice key backend development concepts. Note that these are not the exact questions they will ask in interviews, but they represent the type and pattern commonly expected during technical evaluations or viva rounds.

🧠 *You are encouraged to practice a few questions daily to build confidence and improve your problem-solving approach.*

Feel free to reach out if you need help understanding any concept or want to discuss solutions.

API Sample Interview Questions

1. **Create a GET API to return a list of students.**
2. **Create a POST API to add a new book.**
3. **Create a GET API with a route parameter to fetch a student by ID.**
4. **Create a GET API with query parameters to filter/search data.**
5. **Create a POST API to accept a JSON object and return a confirmation.**
6. **How do you extract data from `req.body` in Express?**
7. **How do you handle 404 errors in an Express app?**
8. **What middleware is used to parse JSON data in Express?**
9. **Explain the difference between `params` and `query` in Express routes.**
10. **Write a route that returns “Hello, World!” when someone visits `/`.**

---

11. **Create an Express app with two routes: `/about` and `/contact`.**
12. **Create a PUT API to update a user's email.**
13. **Create a DELETE API to remove a student by ID.**
14. **Write an Express route that logs each request method and URL.**
15. **Add middleware to validate that `name` and `age` are present in the POST body.**

---

16. **How would you protect an API using a token (JWT)?**

17. **What is the purpose of an Authorization header in an API call?**

18. **How would you verify a token inside Express middleware?**

---

19. **What is the difference between `app.get()` and `app.use()`?**

20. **What are some common use cases for middleware in Express?**

21. **Why is Express.js preferred over just Node.js for building APIs?**

22. **What are HTTP status codes? Give examples (e.g., 200, 404, 500).**

23. **What happens if you don't call `next()` in a middleware?**

1. What is MongoDB and why is it used?
→ MongoDB is a NoSQL database that stores data as JSON-like documents. It's flexible, schema-less, and ideal for fast development with Node.js.

---

2. How do you connect Node.js to MongoDB?
→ Using the `mongoose` or native `mongodb` driver.
Example with Mongoose:

```
mongoose.connect('mongodb://localhost:27017/mydb')
```

---

3. What is Mongoose?
→ Mongoose is an ODM (Object Data Modeling) library for MongoDB. It provides schemas, models, and easy-to-use functions to interact with the database.

---

4. What is a Schema and Model in Mongoose?
→ Schema defines the structure of a document (fields and types). Model is a class that lets you create and query documents using that schema.

5. How do you insert a new document into MongoDB using Mongoose?
→ Create a new model instance and call `.save()`:

```
const student = new Student({ name: 'Ankit', age: 20 });

await student.save();
```

6. How do you read/fetch data using Mongoose?
→ Use methods like:

```
Student.find()

Student.findById(id)

Student.findOne({ name: 'Ankit' })
```

7. How do you update a document in MongoDB using Mongoose?

```
await Student.findByIdAndUpdate(id, { age: 21 });
```

8. How do you delete a document in MongoDB using Mongoose?

```
await Student.findByIdAndDelete(id);
```

🛠️ Practical API-Based Questions (With MongoDB Focus)

9. Create a POST API to add a new user to the database.

→ Assume fields: `name`, `email`, `password`.

---

10. Create a GET API to return all books from the `books` collection.

→ Should return a list of books as JSON.

---

11. Create a PUT API to update a student's marks by ID.

→ Assume route: `/students/:id`, field to update: `marks`.

---

12. Create a DELETE API to remove a book by ID.

→ Route: `/books/:id`.

---

13. Create a GET API to fetch a single user by email using query parameter.

→ Route: `/users?email=abc@gmail.com`.

---

14. Create a POST API that checks if an email is already registered before saving a user.

→ Useful for signup logic.

---

15. Create a GET API to return all students sorted by marks (high to low).

→ Use `.sort({ marks: -1 })` in Mongoose.

---

16. Create an API to return only users who are active.

→ Assume there's a field `isActive: true/false`.

---

17. Create an API to count how many users are registered.

→ Use `User.countDocuments()`.

---

18. Create an API to return only selected fields (name, email) from users.

→ Use `.select('name email')`.

---

19. Create an API to fetch all books written by a specific author.

→ Use route: `/books?author=xyz`.

---

20. Create a POST login API that checks email and password and returns a token.

→ Used in auth-based systems.

---

21. Create middleware that checks if a valid token is sent before allowing access to protected routes.

→ For routes like `/profile`, `/admin`, etc.

🔄 What is a REST API?

REST stands for **Representational State Transfer**. It's a way of designing web services so that different systems (like frontend apps, mobile apps, Postman, etc.) can talk to your server using simple **HTTP methods**.

A **REST API** is an interface that allows clients to access and manipulate data using standard HTTP methods like:

- **GET** – to read data
- **POST** – to create new data
- **PUT** – to update data
- **DELETE** – to delete data

REST APIs are stateless and follow standard URL and method conventions.

🧠 How REST API looks in Express:

```
// GET - Fetch all students
app.get('/students', (req, res) => {
  // logic here
});

// POST - Add new student
app.post('/students', (req, res) => {
  // logic here
});

// PUT - Update entire student data by ID
app.put('/students/:id', (req, res) => {
  // logic here
});
```

```
// DELETE - Delete a student
app.delete('/students/:id', (req, res) => {
  // logic here
});
```

- Can GET requests contain a body? Why or why not?
- Give one real-world example for each HTTP method: **GET**, **POST**, **PUT**, and **DELETE**.
- Name a tool that can be used to test REST APIs and send different HTTP requests manually.