Express.js and Mongodb

Express.js :

1. What is ExpressJS?

ExpressJS is a web application framework for **Node.js**. It helps us build server-side (backend) applications easily using JavaScript.

2. Why use ExpressJS?

We use ExpressJS because it makes building APIs and web servers faster. It simplifies many tasks like routing, handling requests, and managing middleware.

3. Write a 'Hello World' ExpressJS application.

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

4. Differentiate between NodeJS and ExpressJS.

- **Node.js** is a runtime that runs JavaScript outside the browser.
- **Express.js** is a framework that runs on top of Node.js to make building web servers easier.
  In short, Express uses Node, but Node doesn't depend on Express

5. Is ExpressJS a front-end or a back-end framework?

ExpressJS is a **back-end** framework. It runs on the server and handles API routes, databases, authentication, etc.

6. Mention a few features of ExpressJS.

- Fast and lightweight
- Easy routing system
- Middleware support
- REST API support
- Works well with databases

- Uses JavaScript for server-side

7. Explain the structure of an ExpressJS application.

A basic Express project usually has:

- `app.js` or `server.js` (main file)
- `routes/` (route handlers)
- `controllers/` (logic behind routes)
- `models/` (for database schemas)
- `public/` (static files)
- `views/` (for templating, if used)

8. What are some popular alternatives to ExpressJS?

- **Koa.js** (also by the same team as Express)
- **Fastify** (fast and low overhead)
- **NestJS** (uses TypeScript, more structured)
- **Hapi.js** (robust and powerful plugin system)

9. Which major tools can be integrated with ExpressJS?

- **MongoDB** (for database)
- **Mongoose** (ODM for MongoDB)
- **Passport.js** (for authentication)
- **JWT** (for secure tokens)
- **Socket.io** (for real-time apps)
- **Bcrypt** (for password hashing)

10. What is the `.env` file used for?

The `.env` file is used to store **secret keys, API credentials**, and other environment-specific settings. This helps keep sensitive data out of the main code.

---

11. What are JWT?

JWT stands for **JSON Web Token**. It's a secure way to send user identity or authentication data between the client and server.

12. Create a simple middleware for validating user.

```
function validateUser(req, res, next) {
  if (req.user) {
    next();
  } else {
    res.status(401).send('Unauthorized');
  }
}
```

13. What is Bcrypt used for?

Bcrypt is used to **hash passwords** so they are not saved as plain text. It adds security by making the passwords unreadable even if someone gets access to the database.

14. Why should you separate the Express app and server?

Separating them helps in **testing, scaling, and better structure**. The app logic and server can run independently, which makes the project more organized.

15. What do you understand about ESLint?

ESLint is a **code linter tool** that checks your JavaScript code for errors, bad practices, or style issues. It helps you write clean and bug-free code.

16. Define the concept of the test pyramid.

The test pyramid is a model for software testing.

- **Base**: Unit tests (fast and many)
- **Middle**: Integration tests
- **Top**: End-to-end or UI tests (few and slow)

17. Differentiate between `res.send()` and `res.json()`

- `res.send()` can send strings, objects, or buffers.
- `res.json()` only sends JSON, and it automatically sets the `Content-Type` to `application/json`.

18. What is meant by Scaffolding in ExpressJS?

Scaffolding means **auto-generating basic project structure** (like folders, routes, views). It saves time and helps follow a standard format.

19. How would you install an Express application generator for scaffolding?

```
npm install -g express-generator
```

20. What is CORS in ExpressJS?

CORS stands for **Cross-Origin Resource Sharing**. It controls which domains can access your Express server APIs.

21. What are Built-in Middlewares?

Express has built-in middlewares like:

- `express.static()` for serving static files
- `express.json()` to parse JSON bodies
- `express.urlencoded()` to parse form data

22. How would you configure properties in ExpressJS?

You can use `app.set()` and `app.get()` to set and read properties. Example:

```
app.set('view engine', 'ejs');
```

23. Which template engines does Express support?

Express supports engines like:

- **EJS**
- **Pug (Jade)**
- **Handlebars**
  They allow us to create dynamic HTML pages from server-side.

24. Elaborate on various methods of debugging on both Linux and Windows systems.

- Use `console.log()` or `console.error()`
- Use **Node.js Inspector** (`node --inspect`)

- Use **debug module** in Express
- On Linux: use terminal tools like `grep`, `tail -f`, etc.
- On Windows: you can use Visual Studio Code debugger or `nodemon` for auto-reloading and tracking errors

25. Name some databases that integrate with ExpressJS.

- **MongoDB** (with Mongoose)
- **MySQL** (with Sequelize or Knex)
- **PostgreSQL**
- **Firebase**
- **SQLite**

26. How would you render plain HTML using ExpressJS?

You can use:

```
app.use(express.static('public'));
```

Or manually send HTML:

```
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
```

27. What is the use of `response.cookie()` function?

It sets a cookie in the browser. Example:

```
res.cookie('token', '123abc', { httpOnly: true });
```

28. Under what circumstances does a Cross-Origin request fail in ExpressJS?

CORS can fail if:

- The frontend and backend are on different domains and **CORS is not enabled**
- The request is not allowed in the **CORS settings**
- Wrong headers or methods are used (like PUT, DELETE, etc.)

MongoDb :

1. What is MongoDB, and How Does It Differ from Traditional SQL Databases?

MongoDB is a **NoSQL database** that stores data in **JSON-like documents**. Unlike SQL databases (like MySQL), MongoDB doesn't use tables or rows—it uses **collections and documents**, which makes it flexible for unstructured data.

2. Explain BSON and Its Significance in MongoDB.

BSON (Binary JSON) is a format used by MongoDB to store documents. It's similar to JSON but supports extra types like `Date`, `ObjectId`, etc. It's **faster** and more **efficient for storage and parsing**.

3. Describe the Structure of a MongoDB Document.

A MongoDB document is a **key-value pair** structure, much like a JavaScript object.
Example:

```
{
  "_id": 1,
  "name": "Alice",
  "age": 25,
  "skills": ["JS", "React"]
}
```

4. What are Collections and Databases in MongoDB?

- A **database** is a container for collections.
- A **collection** holds related documents (like a table in SQL).
  They help organize and group your data.

5. How Does MongoDB Ensure High Availability and Scalability?

MongoDB supports:

- **Replica sets** for high availability (auto failover)
- **Sharding** for horizontal scalability (splitting data across servers)

6. Explain the Concept of Replica Sets in MongoDB.

A **replica set** is a group of MongoDB servers where one is **primary** and others are **secondary**. If the primary fails, one secondary automatically becomes the new primary.

7. What are the Advantages of Using MongoDB Over Other Databases?

- Flexible schema
- Easy to scale
- Fast for big data
- Stores nested data
- Developer-friendly (uses JSON-like syntax)

8. How to Create a New Database and Collection in MongoDB?

In the shell:

```
use myDatabase

db.createCollection("users")
```

9. What is Sharding, and How Does It Work in MongoDB?

Sharding is splitting large data into smaller parts called **shards**, stored on different servers. MongoDB uses a **shard key** to decide how to distribute data. It helps manage **big datasets efficiently**.

10. Explain the Basic Syntax of MongoDB CRUD Operations.

- **Create**: `db.users.insertOne({ name: "Ali" })`
- **Read**: `db.users.find({ name: "Ali" })`
- **Update**: `db.users.updateOne({ name: "Ali" }, { $set: { age: 30 } })`
- **Delete**: `db.users.deleteOne({ name: "Ali" })`

11. How to Perform Basic Querying in MongoDB?

Example queries:

```
db.users.find({ age: { $gt: 18 } })       // greater than

db.users.find({ name: /Ali/i })           // regex

db.users.find({ $or: [{ age: 20 }, { age: 25 }] })
```

12. What is an Index in MongoDB, and How to Create One?

An **index** speeds up searching in collections.
Create one with:

```
db.users.createIndex({ name: 1 })  // 1 for ascending
```

13. How Does MongoDB Handle Data Consistency?

MongoDB uses **write concern** and **read concern** to control consistency. Also, with **replica sets**, writes go to the primary, ensuring consistent data before syncing to secondaries.

14. How to Perform Data Import and Export in MongoDB?

- Import:
  ```
  mongoimport --db test --collection users --file users.json
  ```

- Export:
  ```
  mongoexport --db test --collection users --out users.json
  ```

15. What are MongoDB Aggregation Pipelines and How are They Used?

Aggregation pipelines allow data processing in stages like filtering, grouping, and sorting.
Example:

```
db.orders.aggregate([

  { $match: { status: "delivered" } },

  { $group: { _id: "$customer", total: { $sum: "$amount" } } }

])
```