Node.JS

Assignment-2

Q.1. Explain the role of middleware in Express.JS and demonstrate how to implement a custom middleware function that logs the details of incoming HTTP requests.

⇒ Middleware in Express.js are functions that execute between the request and the response. They can:

- Access req and res objects.
- Execute code, modify data, or stop the request.
- Call next() to move the next middleware

Use of middleware:

- Logging requests.
- Authentication and authorization
- Errors handling
- parsing JSON or form data.

Example: custom logger middleware

```
const express = require('express');
const app = express();
//custom middleware
function logger(req, res, next){
    console.log(`${req.method} ${req.url}`);
    next(); // pass control to next middleware
}
app.use(logger);
app.get('/', (req, res) => res.send('Hello, Express!'));
app.listen(3000, () => console.log("Server running on 3000"));
```

Q.2. Describe the process of setting up authentication using JSON web Token (JWT) in a Node.js application. Include key steps such as generating tokens, verify tokens, and handling user sessions.

⇒ JWT (JSON Web Token) is used for secure, stateless authentication in APIs

Steps:-

(i) User Login: User provides email/password.

(ii) Verify user: Check credentials from the database.

(iii) Generate Token: If valid, create a JWT using jwt.sign().

(iv) Send Token: Return it to the client.

(v) Client sends Token: On every request, client sends Authorization: Bearer <token>.

(vi) Verify Token: Middleware verifies token using jwt.verify().

Example:-

```
const token = jwt.sign({ id: user.id }, 'secret',
                { expiresIn: '1h' });
```

Q.3. Discuss the importance of securing routes and resources in an Express.js application. Describe the implementation of role base access control (RBAC) and give an example of how to restrict access to specific routes based on user roles.

⇒ Importance of security:
- Prevents unauthorized data access.
- Protects admin-only operations.
- Keeps sensitive APIs safe.

RBAC (Role Base Access Control):
- Each user has a roole (eg. user, admin).
- Routes check if users role is allowed.

Example:

```
function allowRoles (...roles) {
  return (req, res, next) => {
    if (!roles.includes (req.user.role)) return res.status
        (403).send ('Forbidden');
    next ();
  };
}
```

Q.4. Describe the RESTful architecture and explain how to setup a basic Express.js servers to handle CRUD operations for a resource (eg. "users"). Include examples of route definitions, HTTP request methods, and handling responses.

⇒ REST = Representational State Transfers (use HTTP methods for operations).

CRUD routes:
- GET → Read
- POST → Create
- PUT → Update
- DELETE → Delete

Example:
app.get('/users', ...);
app.post('/users', ...);
app.put('/users/:id', ...);
app.delete('/users/:id', ...);

Q.5. Explain the concept of authentication and authorization in the context of a Node.js application. Discuss the differences between local authentication using Passport and token-based authentication using JWT. Provide a detailed example of implementing local authentication with Passport.

⇒ Authentication → Confirms the users identity (login).
Authorization → Controls what the user can access (permissions)

Difference:

Passport (local)	JWT
(i) Uses session and cookies	(i) Uses tokens
(ii) Server stores session	(ii) Stateless (no session)
(iii) Good for web apps	(iii) Best for APIs / mobile.

Local (passport): Uses session & cookies
JWT: Stateless, token-based.
Passport example:
passport.use(new LocalStrategy(...));
app.post('/login', passport.authenticate('local'), (req, res) ⇒ res.send("Logged in"));

Q.6. Discuss the various database options available for Node.js applications, specifically focusing on MongoDB, MySQL, and PostgreSQL. Explain how to connect a Node.js application to a MongoDB database using Mongoose, define a schema, and perform basic CRUD operation. Include code snippets to illustrate your points.

⇒ Database options:
* MongoDB: NoSQL, document-based, flexible schema.
* MySQL: Relational, table-based, easy to use.
* PostgreSQL: Relational, advanced features (triggers, JSON support).

Connect + CRUD (Mongoose):

```
mongoose.connect('mongodb://127.0.0.1:27017/testdb');
const User = mongoose.model('User', new mongoose.Schema
    ({name: String}));
await User.create({name 'John'});
await User.find();
await User.updateOne({name: 'John'},
    {name: 'John'});
await User.deleteOne({name: 'Jonny'});
```