📘 Full Stack Web Development-II (Practical Solutions)

Course Code: 05010105DS15
Semester: 5
Subject: Full Stack Web Development-II (P)

---

1. Node.js program that reads a user's name asynchronously

Steps to Perform:

- Create `greet.js`
- Use `readline` module
- Print greeting asynchronously with `setTimeout`
- Run: `node greet.js`

Solution:

```
const readline = require("readline");
const rl = readline.createInterface({ input: process.stdin, output: process.stdout });

rl.question("Enter your name: ", (name) => {
  setTimeout(() => {
    console.log(`Hello, ${name}! Welcome to Node.js`);
    rl.close();
  }, 1000);
});
```

Expected Output:

```
Enter your name: John
Hello, John! Welcome to Node.js
```

---

2. CRUD on text file using fs module

Steps to Perform:

- Create fileCRUD.js
- Use fs module
- Perform create, read, update, delete

Solution:

```
const fs = require("fs");
fs.writeFileSync("example.txt", "Hello, this is first content!");
console.log("File Content:", fs.readFileSync("example.txt", "utf-8"));
fs.appendFileSync("example.txt", "\nThis is appended text.");
console.log("Updated:", fs.readFileSync("example.txt", "utf-8"));
fs.unlinkSync("example.txt");
console.log("File deleted.");
```

Expected Output:

```
File Content: Hello, this is first content!
Updated: Hello, this is first content!
This is appended text.
File deleted.
```

---

3. Basic Express.js Server

Steps to Perform:

- Install express
- Create server.js
- Run node server.js

Solution:

```
const express = require("express");
const app = express();
const PORT = 3000;
```

```
app.get("/", (req, res) => res.send("Hello, Express!"));
app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

Expected Output:

```
Server running on http://localhost:3000

Browser → Hello, Express!
```

---

4. Express.js with GET & POST

Steps to Perform:

- Extend Express app
- Use express.json()
- Add routes

Solution:

```
const express = require("express");
const app = express();
app.use(express.json());

app.get("/user", (req, res) => res.json({ name: "Alice", age: 22 }));
app.post("/user", (req, res) => res.json({ msg: "User data received", data: req.body
}));
app.listen(3000, () => console.log("Server running at http://localhost:3000"));
```

Expected Output:

```
GET /user → {"name":"Alice","age":22}
POST /user { "name": "Bob" } → {"msg":"User data
received","data":{"name":"Bob"}}
```

---

5. RESTful User Management API

Steps to Perform:

- Define CRUD routes

- Use in-memory array

Solution:

```
const express = require("express");
const app = express();
app.use(express.json());
let users = [{ id: 1, name: "Alice" }];

app.get("/users", (req, res) => res.json(users));
app.post("/users", (req, res) => { const user = { id: Date.now(), ...req.body };
users.push(user); res.json(user); });
app.put("/users/:id", (req, res) => { const id = parseInt(req.params.id); users =
users.map(u => (u.id === id ? { ...u, ...req.body } : u)); res.json({ msg: "Updated" });
});
app.delete("/users/:id", (req, res) => { users = users.filter(u => u.id !==
parseInt(req.params.id)); res.json({ msg: "Deleted" }); });

app.listen(3000, () => console.log("User API running..."));
```

Expected Output:

```
GET /users → [{id:1,name:'Alice'}]
POST /users {name:'Bob'} → {id:...,name:'Bob'}
```

---

6. Real-Time Chat with Socket.io

Steps to Perform:

- Install express & socket.io
- Create server
- Handle chat messages

Solution:

```
const express = require("express");
const http = require("http");
const { Server } = require("socket.io");
const app = express();
const server = http.createServer(app);
const io = new Server(server);

io.on("connection", (socket) => {
  console.log("User connected");
  socket.on("chat message", (msg) => io.emit("chat message", msg));
});

app.get("/", (req, res) => res.sendFile(__dirname + "/index.html"));
server.listen(3000, () => console.log("Chat running at http://localhost:3000"));
```

Expected Output:

- Open in 2 browsers → messages broadcast live

---

7. Node.js + MongoDB CRUD

Steps to Perform:

- Install mongodb driver

- Connect and perform CRUD

Solution:

```
const { MongoClient } = require("mongodb");
const url = "mongodb://localhost:27017";
const client = new MongoClient(url);

async function run() {
  await client.connect();
  const db = client.db("testdb");
  const users = db.collection("users");

  await users.insertOne({ name: "Alice", age: 22 });
  console.log(await users.find().toArray());
  await users.updateOne({ name: "Alice" }, { $set: { age: 23 } });
  console.log(await users.findOne({ name: "Alice" }));
  await users.deleteOne({ name: "Alice" });
  console.log("Deleted");
  await client.close();
}
```

```
run();
```

Expected Output:

```
[ { _id: ..., name: 'Alice', age: 22 } ]
{ _id: ..., name: 'Alice', age: 23 }
Deleted
```

---

8. Express API + MongoDB CRUD

Steps to Perform:

- Install mongoose
- Connect DB
- Define routes

Solution:

```
const express = require("express");
const mongoose = require("mongoose");
const app = express();
app.use(express.json());
mongoose.connect("mongodb://localhost:27017/testdb");

const User = mongoose.model("User", { name: String, age: Number });
app.get("/users", async (req, res) => res.json(await User.find()));
app.post("/users", async (req, res) => res.json(await User.create(req.body)));

app.listen(3000, () => console.log("Mongo API running"));
```

Expected Output:

```
GET /users → []
POST /users {name:'Alice', age:22} → { _id:'...', name:'Alice', age:22 }
```

---

9. RESTful API with Express + MongoDB

⚡ Steps to Perform

Initialize Project

mkdir rest-api

cd rest-api

npm init -y

npm install express mongoose

1.
2. **Setup Express Server in `server.js`.**
3. **Connect MongoDB (local or MongoDB Atlas).**
4. **Create Model (e.g., `User`).**
5. **Define RESTful Routes:**
    - `GET /users` **→ fetch all users**
    - `GET /users/:id` **→ fetch single user**
    - `POST /users` **→ create user**
    - `PUT /users/:id` **→ update user**
    - `DELETE /users/:id` **→ delete user**

Run API

node server.js

6. **Test endpoints using Postman / curl.**

📝 Full Code: `server.js`

const express = require("express");

```
const mongoose = require("mongoose");


const app = express();

app.use(express.json()); // to parse JSON bodies


// 1. Connect to MongoDB

mongoose.connect("mongodb://localhost:27017/restapidb")

  .then(() => console.log("MongoDB connected"))

  .catch(err => console.error(err));


// 2. Create Schema & Model

const userSchema = new mongoose.Schema({

  name: String,

  email: String,

  age: Number

});


const User = mongoose.model("User", userSchema);


// 3. Routes


// GET all users

app.get("/users", async (req, res) => {

  const users = await User.find();
  res.json(users);
});

// GET single user
app.get("/users/:id", async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) return res.status(404).json({ msg: "User not found" });
    res.json(user);
  } catch (err) {
    res.status(400).json({ msg: "Invalid ID" });
  }
});

// POST create user
app.post("/users", async (req, res) => {
  const user = new User(req.body);
  await user.save();
  res.status(201).json(user);
});

// PUT update user
app.put("/users/:id", async (req, res) => {
  try {
    const updated = await User.findByIdAndUpdate(req.params.id, req.body, { new:
true });

    if (!updated) return res.status(404).json({ msg: "User not found" });

    res.json(updated);

  } catch (err) {

    res.status(400).json({ msg: "Invalid ID" });

  }

});


// DELETE remove user

app.delete("/users/:id", async (req, res) => {

  try {

    const deleted = await User.findByIdAndDelete(req.params.id);

    if (!deleted) return res.status(404).json({ msg: "User not found" });

    res.json({ msg: "User deleted" });

  } catch (err) {

    res.status(400).json({ msg: "Invalid ID" });

  }

});


// 4. Start Server

const PORT = 3000;

app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

🎯 Sample Output

POST /users

```
 {
  "_id": "64f...",
  "name": "Alice",
  "email": "alice@example.com",
  "age": 22,
  "__v": 0
}
```

-

GET /users

```
 [
  {
    "_id": "64f...",
    "name": "Alice",
    "email": "alice@example.com",
    "age": 22
  }
]
```

-

PUT /users/64f...

```
 {
  "_id": "64f...",
```

  "name": "Alice Johnson",

  "email": "alice@example.com",

  "age": 23

}

-

DELETE /users/64f...

 { "msg": "User deleted" }

-

---

10. Node.js Async Greeting (Duplicate)

⚡ Steps to Perform

1. Create a new file, e.g. `greet.js`.
2. Use Node.js `readline` module to take user input.
3. Use `setTimeout` to print the message asynchronously.

Run the program:

```
node greet.js
```

4.

📝 Code: `greet.js`

```
// Import readline module
const readline = require("readline");

// Create interface for input/output
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// Ask user for their name
rl.question("Enter your name: ", (name) => {
  // Asynchronous greeting after 1 second
  setTimeout(() => {
    console.log(`Hello, ${name}! Welcome to Node.js`);
    rl.close();
  }, 1000);
});
```

🎯 Example Run

```
Enter your name: John
Hello, John! Welcome to Node.js
```

11. JWT Authentication in Express.js

Steps to Perform:

- **Install express & jsonwebtoken**
- **Create login & protected routes**

Solution:

```
const express = require("express");
const jwt = require("jsonwebtoken");
const app = express();
app.use(express.json());

const SECRET = "secret123";
app.post("/login", (req, res) => {
  const token = jwt.sign({ user: req.body.username }, SECRET, { expiresIn: "1h" });
  res.json({ token });
});

app.get("/protected", (req, res) => {
  const auth = req.headers.authorization?.split(" ")[1];
  try {
    const decoded = jwt.verify(auth, SECRET);
    res.json({ msg: "Protected data", user: decoded.user });
  } catch {
    res.status(401).json({ msg: "Unauthorized" });
  }
});

app.listen(3000, () => console.log("JWT app running"));
```

Expected Output:

```
POST /login → { "token": "..." }
GET /protected (with token) → { "msg":"Protected data","user":"Alice" }
```

12 RESTful API: User Registration & Login

⚡ Steps to Perform

Initialize Project

mkdir auth-api

cd auth-api

npm init -y

npm install express mongoose bcryptjs jsonwebtoken

1.
2. Create Server (`server.js`).
3. Connect MongoDB (local or Atlas).
4. Create User model with `name`, `email`, `password`.
5. Hash password before saving using `bcryptjs`.
6. Generate JWT on login using `jsonwebtoken`.
7. Expose Routes:
    - `POST /register` → register new user
    - `POST /login` → login and return JWT

📝 Full Code: `server.js`

const express = require("express");

const mongoose = require("mongoose");

const bcrypt = require("bcryptjs");

const jwt = require("jsonwebtoken");

```
const app = express();
app.use(express.json());

const SECRET = "mysecret"; // put in .env for real apps

// 1. Connect MongoDB
mongoose.connect("mongodb://localhost:27017/authdb")
  .then(() => console.log("MongoDB connected"))
  .catch(err => console.error(err));

// 2. Define Schema & Model
const userSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  password: String
});

const User = mongoose.model("User", userSchema);

// 3. Register Route
app.post("/register", async (req, res) => {
  try {
    const { name, email, password } = req.body;
    // check if email exists
    const existing = await User.findOne({ email });
    if (existing) return res.status(400).json({ msg: "Email already registered" });

    // hash password
    const hashed = await bcrypt.hash(password, 10);

    const user = new User({ name, email, password: hashed });
    await user.save();
    res.status(201).json({ msg: "User registered successfully" });
  } catch (err) {
    res.status(500).json({ msg: "Server error", error: err.message });
  }
});

// 4. Login Route
app.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ msg: "Invalid email or password" });
    const match = await bcrypt.compare(password, user.password);

    if (!match) return res.status(400).json({ msg: "Invalid email or password" });


    // generate JWT

    const token = jwt.sign({ id: user._id, email: user.email }, SECRET, { expiresIn: 
"1h" });


    res.json({ msg: "Login successful", token });

  } catch (err) {

    res.status(500).json({ msg: "Server error", error: err.message });

  }

});


// 5. Start Server

const PORT = 3000;

app.listen(PORT, () => console.log(`Auth API running on 
http://localhost:${PORT}`));
```

Outputs

Register

POST /register

{ "name": "Alice", "email": "alice@example.com", "password": "123456" }

→ { "msg": "User registered successfully" }

-

Login

POST /login

{ "email": "alice@example.com", "password": "123456" }

→ { "msg": "Login successful", "token": "eyJhbGciOi..." }

-

13. Dockerize Node.js App

Steps to Perform:

- Create Dockerfile

- Build and run image

Solution (Dockerfile):

```
FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

Expected Output:

Server runs inside Docker container at localhost:3000

14. Blog REST API (Express.js)

Steps to Perform:

- Create Express routes

- **Use in-memory posts**

Solution:

```
const express = require("express");
const app = express();
app.use(express.json());
let posts = [];

app.get("/posts", (req, res) => res.json(posts));
app.post("/posts", (req, res) => {
  const post = { id: Date.now(), ...req.body };
  posts.push(post);
  res.json(post);
});

app.listen(3000, () => console.log("Blog API running"));
```

Expected Output:

```
POST /posts {title:"First", body:"Hello"} → {id:..., title:"First", body:"Hello"}
```

---

15. Passport.js Authentication

Steps to Perform:

- **Install passport, passport-local, express-session**

- **Configure Local Strategy**

Solution:

```
const express = require("express");
const passport = require("passport");
const LocalStrategy = require("passport-local").Strategy;
const session = require("express-session");

const app = express();
app.use(express.urlencoded({ extended: false }));
app.use(session({ secret: "secret" }));
app.use(passport.initialize());
app.use(passport.session());
```

```
passport.use(new LocalStrategy((username, password, done) => {
  if (username === "admin" && password === "1234") return done(null, {
username });
  return done(null, false);
}));
passport.serializeUser((user, done) => done(null, user));
passport.deserializeUser((user, done) => done(null, user));

app.post("/login", passport.authenticate("local"), (req, res) => res.send("Logged
in"));
app.listen(3000, () => console.log("Passport auth running"));
```

Expected Output:

```
POST /login {username:"admin",password:"1234"} → "Logged in"
```
