

Batch: A1

Roll No.: 16010123012

Experiment / assignment / tutorial No.: 07

Grade: AA / AB / BB / BC / CC / CD / DD

Signature of the Staff In-charge with date

Title: Create a RESTful API server in Express and Node.js. Implementation + Testing application using postman/Thunderclient

AIM: Create a RESTful API server in Express and Node.js. Implementation + Testing application using postman/Thunderclient

Problem Definition:

Develop a comprehensive RESTful API server for ThermoAQ (Thermal Air Quality) application that provides endpoints for:

- User authentication and authorization
- Weather and Air Quality Index (AQI) data retrieval
- User profile management
- Location-based monitoring

Resources used:

<https://learning.postman.com/>

<https://expressjs.com/en/resources/middleware.html>

Expected OUTCOME of Experiment:

CO 3: Test the concepts and components of various front-end, back-end web app development technologies & frameworks using web development tools.

Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2 nd Edition, 2016.

Pre Lab/ Prior Concepts:

Write details about the following content

- **Testing in POSTMAN** - Postman is an API development and testing platform that simplifies the process of building, testing, and documenting APIs.
 1. HTTP Methods

- `GET` - Retrieve data from server
- `POST` - Create new resources
- `PUT` - Update existing resources
- `DELETE` - Remove resources
- `PATCH` - Partial updates

2. Request Components

- URL/Endpoint - API route (e.g., `http://localhost:3001/api/auth/login`)
- Headers - Metadata (Content-Type, Authorization)
- Body - Request payload (JSON, form-data)
- Query Parameters - URL parameters (?key=value)
- Path Variables - Dynamic route segments (/user/:id)

3. Response Components

- Status Code - HTTP status (200, 404, 500)
- Headers - Response metadata
- Body - Response data (JSON)
- Time - Response time in milliseconds

Methodology:

The experiment was carried out by setting up a RESTful API server using Node.js and Express.js. A MongoDB database was connected using Mongoose to store user information securely. The project followed a modular approach where separate files were created for routes, models, and middleware. User authentication was implemented using JWT (JSON Web Token) and bcrypt for password encryption. Postman was used for testing each API endpoint, verifying correct responses for user signup, login, profile updates, and AQI data retrieval. The system was designed to follow REST principles with proper HTTP methods and status codes.

Implementation Details:

1. Database Schema (MongoDB)

```
// User Model
{
  name: String,
  email: String (unique),
  password: String (hashed),
```

```
preferences: {  
  location: String,  
  coordinates: Object,  
  aqiAlertThreshold: Number,  
  emailNotifications: Boolean,  
  pushNotifications: Boolean  
},  
alerts: [{  
  message: String,  
  severity: String,  
  aqi: Number,  
  location: String,  
  createdAt: Date,  
  read: Boolean  
}],  
createdAt: Date,  
updatedAt: Date  
}
```

2. API Endpoints Implemented

Authentication Routes (`/api/auth`)

```
POST /api/auth/signup    - Register new user  
POST /api/auth/login     - User login  
GET  /api/auth/me        - Get user  
POST /api/auth/logout    - User logout
```

User Routes (`/api/user`) - Protected

```
GET  /api/user/profile    - Get user profile  
PUT  /api/user/profile    - Update user profile  
GET  /api/user/location   - Get user's saved location  
POST /api/user/location   - Update user location  
GET  /api/user/alerts     - Get user alerts  
DELETE /api/user/alerts   - Clear all alerts  
POST /api/user/check-alerts - Trigger alert check  
GET  /api/weather/current?city={city} - Weather & AQI data  
GET  /api/aqi/:city       - Air Quality Index data
```

Steps for execution:

1. Install Dependencies
2. Server.js

```
const express = require('express');  
const mongoose = require('mongoose');  
const cors = require('cors');  
require('dotenv').config();
```

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

mongoose.connect(process.env.MONGODB_URI)
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('MongoDB connection error:', err));

app.use('/api/auth', require('./routes/auth'));
app.use('/api/user', require('./routes/user'));

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

3. Create User Model

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

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  preferences: {
    location: String,
    aqiAlertThreshold: { type: Number, default: 150 }
  },
  alerts: [{
    message: String,
    severity: String,
    createdAt: { type: Date, default: Date.now }
  }]
}, { timestamps: true });

module.exports = mongoose.model('User', userSchema);
```

4. Create auth middleware

```
const jwt = require('jsonwebtoken');

module.exports = async (req, res, next) => {
```

```
try {
  const token = req.header('Authorization')?.replace('Bearer ', '');

  if (!token) {
    return res.status(401).json({ message: 'No token provided' });
  }

  const decoded = jwt.verify(token, process.env.JWT_SECRET);
  req.user = decoded;
  next();
} catch (error) {
  res.status(401).json({ message: 'Invalid token' });
}
};
```

5. Create Routes

```
const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const router = express.Router();

// Signup
router.post('/signup', async (req, res) => {
  try {
    const { name, email, password } = req.body;

    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: 'User already exists' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({
      name,
      email,
      password: hashedPassword
    });

    await user.save();
    const token = jwt.sign(
      { id: user._id },
```

```
        process.env.JWT_SECRET,
        { expiresIn: '7d' }
    );

    res.status(201).json({
        success: true,
        token,
        user: { id: user._id, name, email }
    });
} catch (error) {
    res.status(500).json({ message: 'Server error' });
}
});

// Login
router.post('/login', async (req, res) => {
    try {
        const { email, password } = req.body;

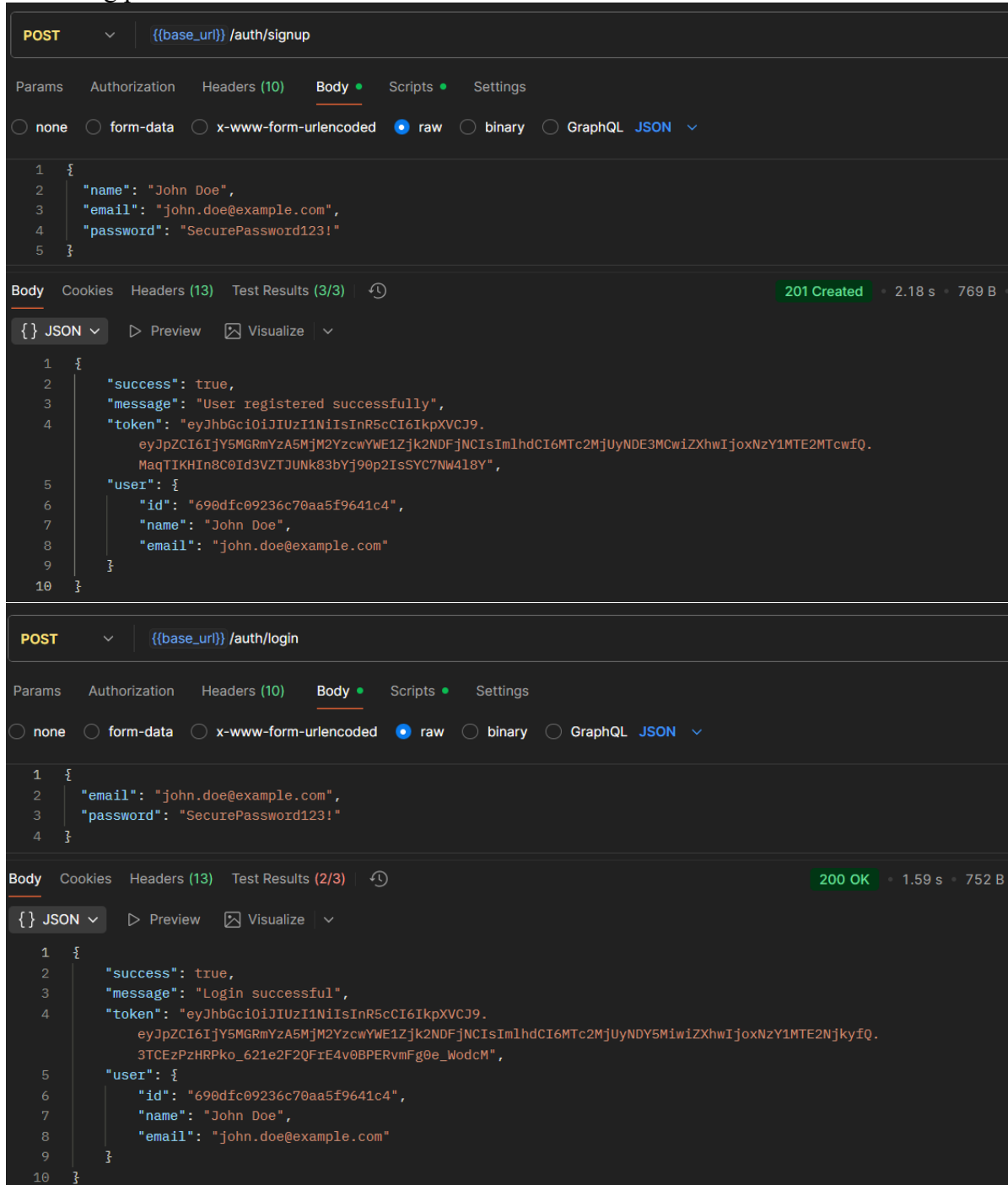
        const user = await User.findOne({ email });
        if (!user) {
            return res.status(400).json({ message: 'Invalid credentials' });
        }
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(400).json({ message: 'Invalid credentials' });
        }

        const token = jwt.sign(
            { id: user._id },
            process.env.JWT_SECRET,
            { expiresIn: '7d' }
        );

        res.json({
            success: true,
            token,
            user: { id: user._id, name: user.name, email: user.email }
        });
    } catch (error) {
        res.status(500).json({ message: 'Server error' });
    }
});
```

```
module.exports = router;
```

6. Start the server
7. Test using postman



GET `{{base_url}} /auth/me`

Params **Authorization** Headers (8) Body Scripts Settings

Auth Type: Bearer Token

Token:

Body Cookies Headers (13) Test Results (2/2) 200 OK • 445 ms • 972 B

JSON Preview Visualize

```

1  {
2    "success": true,
3    "user": {
4      "location": {
5        "city": "Mumbai",
6        "state": "Maharashtra",
7        "latitude": null,
8        "longitude": null,
9        "lastUpdated": "2025-11-07T14:02:49.283Z"
10     },
11     "preferences": {
12       "autoDetectLocation": true,
13       "defaultLocation": "Mumbai, Maharashtra",
14       "aqiAlertThreshold": 150,
15       "enableAlerts": true,
16       "healthConditions": []
17     },
18     "_id": "690dfc09236c70aa5f9641c4",
19     "name": "John Doe",
20     "email": "john.doe@example.com",
21     "monitoredLocations": [],
22     "aqiHistory": [],
23     "alerts": [],
24     "healthReports": [],
25     "createdAt": "2025-11-07T14:02:49.314Z",

```

POST `{{base_url}} /auth/logout`

Params **Authorization** Headers (9) Body Scripts Settings

Auth Type: Bearer Token

Token:

Body Cookies Headers (13) Test Results 200 OK • 92 ms • 485 B

JSON Preview Visualize

```

1  {
2    "success": true,
3    "message": "Logged out successfully"
4  }

```

Conclusion:

The experiment successfully demonstrated the creation of a secure and scalable RESTful API using Express.js and Node.js. It provided practical experience in backend development, authentication mechanisms, and database integration with MongoDB. Testing through Postman verified all endpoints, ensuring reliable communication between client and server. This enhanced understanding of full-stack web development concepts and API lifecycle management.