Worksheet No.: 6

Student Name: Harsh
UID: 24MCA20045
Branch: MCA (General)
Section/Group: 1-A
Semester: 2nd
Date of Performance : 08/04/2025
Subject Name: Advanced internet programming lab
Subject Code: 24CAP-652

1. Aim of the practical:

Create and consume Restful web services for accessing employee data application securely.

2. Apparatus:

Visual Studio code (Vs code), Mongodb, Postman.

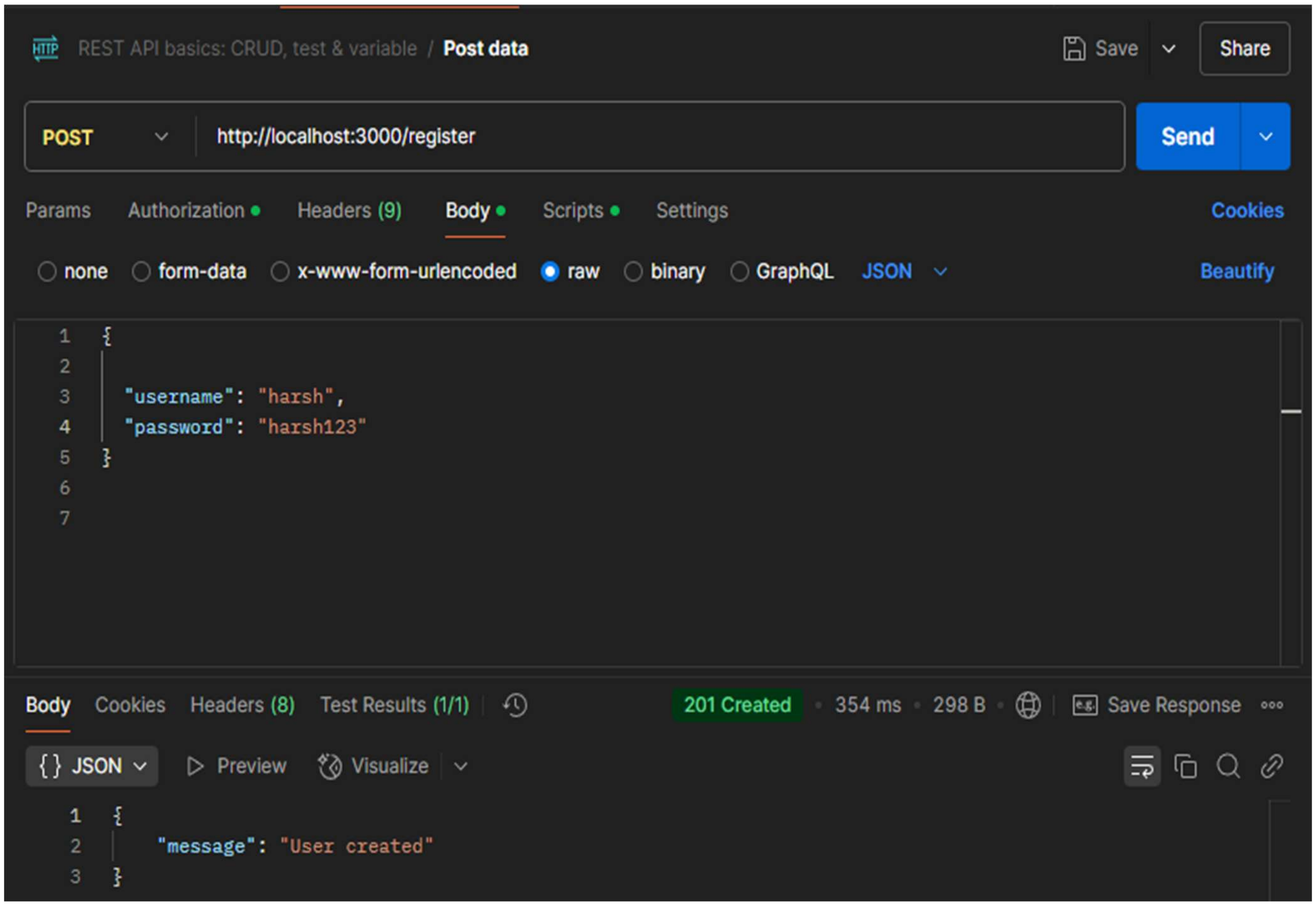
3. Code for experiment/practical:

```
// Import required modules
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const dotenv = require('dotenv');
const cors = require('cors');

dotenv.config();

// Initialize express app
const app = express();

// Use middlewares
app.use(bodyParser.json());
app.use(cors());

// MongoDB Connection
mongoose.connect('mongodb://localhost:27017/Employee', { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch((err) => console.log(err));

// Employee Schema
const employeeSchema = new mongoose.Schema({
  name: { type: String, required: true },
  position: { type: String, required: true },
  salary: { type: Number, required: true },
});

const Employee = mongoose.model('Employee', employeeSchema);

// User Schema for Authentication
const userSchema = new mongoose.Schema({
  username: { type: String, required: true },
  password: { type: String, required: true },
});

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

// Register a new user (Admin)
app.post('/register', async (req, res) => {
  const { username, password } = req.body;
  const hashedPassword = await bcrypt.hash(password, 10);
  const newUser = new User({ username, password: hashedPassword });

  try {
    await newUser.save();
    res.status(201).json({ message: 'User created' });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Login to get a JWT token
```

```
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  const user = await User.findOne({ username });
  if (!user) return res.status(400).json({ message: 'Invalid credentials' });

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

  const token = jwt.sign({ userId: user._id, username: user.username }, 'MasterApp', { expiresIn:
'1h' });
  res.json({ token });
});

// Middleware to authenticate JWT token
const authenticate = (req, res, next) => {
  const token = req.header('Authorization')?.replace('Bearer ', '');
  if (!token) return res.status(401).json({ message: 'Authentication required' });

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

// CRUD operations for Employees

// Create a new employee
app.post('/employees', authenticate, async (req, res) => {
  const { name, position, salary } = req.body;
  const newEmployee = new Employee({ name, position, salary });

  try {
    const savedEmployee = await newEmployee.save();
    res.status(201).json(savedEmployee);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Get all employees
app.get('/employees', authenticate, async (req, res) => {
  try {
    const employees = await Employee.find();
    res.status(200).json(employees);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Get employee by ID
app.get('/employees/:id', authenticate, async (req, res) => {
  try {
    const employee = await Employee.findById(req.params.id);
    if (!employee) return res.status(404).json({ message: 'Employee not found' });
    res.status(200).json(employee);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Update an employee by ID
app.put('/employees/:id', authenticate, async (req, res) => {
  const { name, position, salary } = req.body;
  try {
    const updatedEmployee = await Employee.findByIdAndUpdate(
      req.params.id,
      { name, position, salary },
      { new: true }
    );
    res.status(200).json(updatedEmployee);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Delete an employee by ID
```

```
app.delete('/employees/:id', authenticate, async (req, res) => {
  try {
    await Employee.findByIdAndDelete(req.params.id);
    res.status(200).json({ message: 'Employee deleted' });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Start the server
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

4. **Result/Output/Writing Summary:**

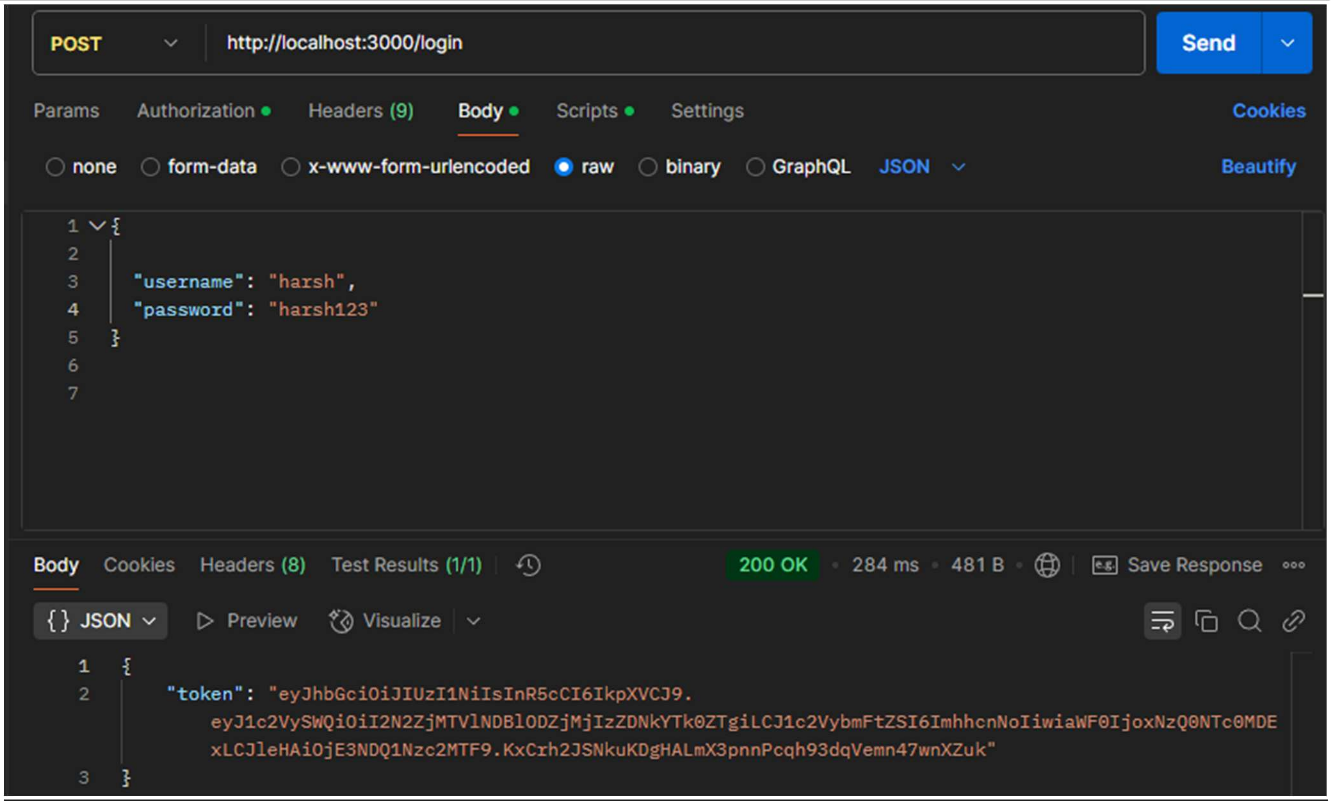


POST http://localhost:3000/register

```
{

  "username": "harsh",
  "password": "harsh123"
}
```

201 Created · 354 ms · 298 B

```
{
  "message": "User created"
}
```

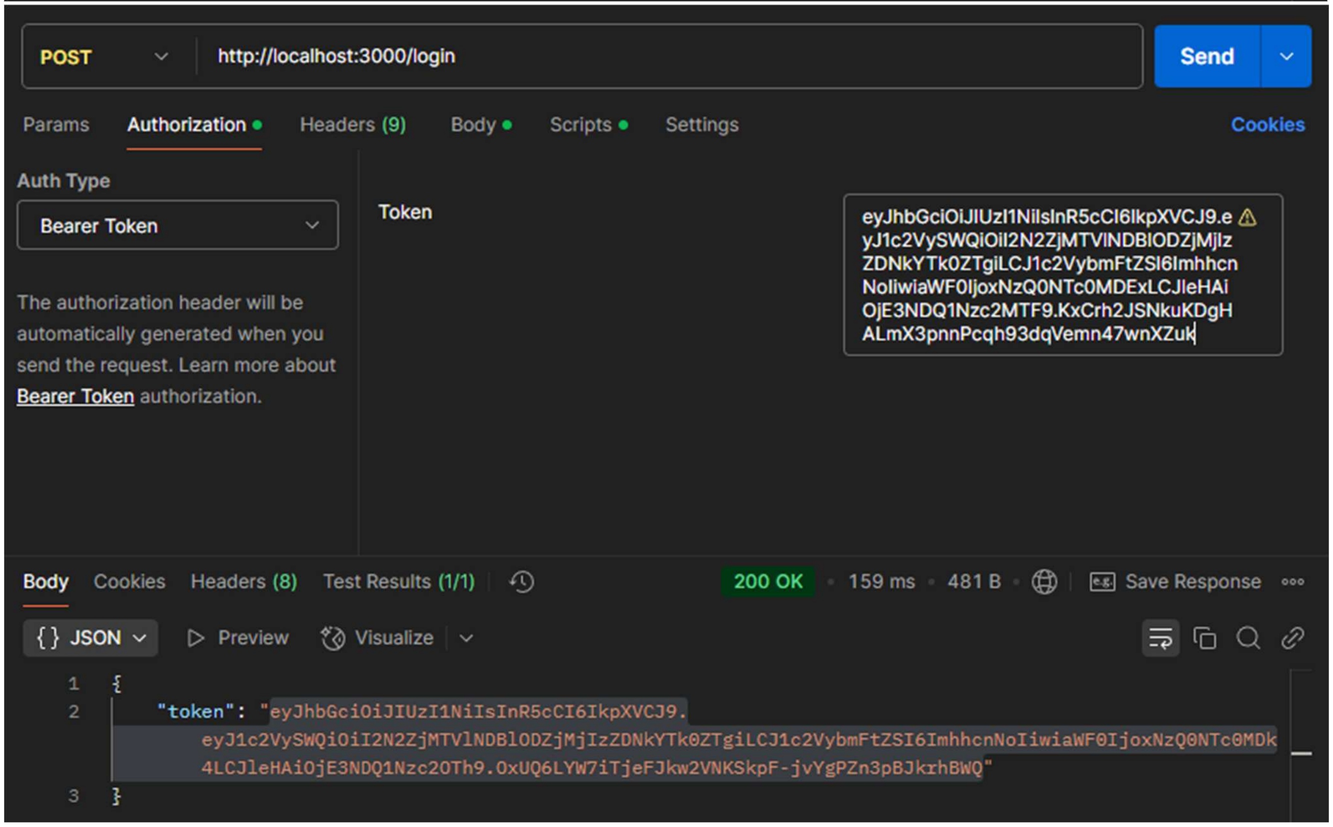


http://localhost:3000/login

```
{

  "username": "harsh",
  "password": "harsh123"
}
```

200 OK · 284 ms · 481 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
      eyJ1c2VySWQiOiI2N2ZjMTVlNDBlODZjMjIzZDNkYTk0ZTgiLCJ1c2VybmFtZSI6ImhhcnNoIiwiaWF0IjoxNzQ0NTc0MDE
      xLCJleHAiOjE3NDQ1Nzc2MTF9.KxCrh2JSNkuKDgHALmX3pnnPcqh93dqVemn47wnXZuk"
}
```



http://localhost:3000/login

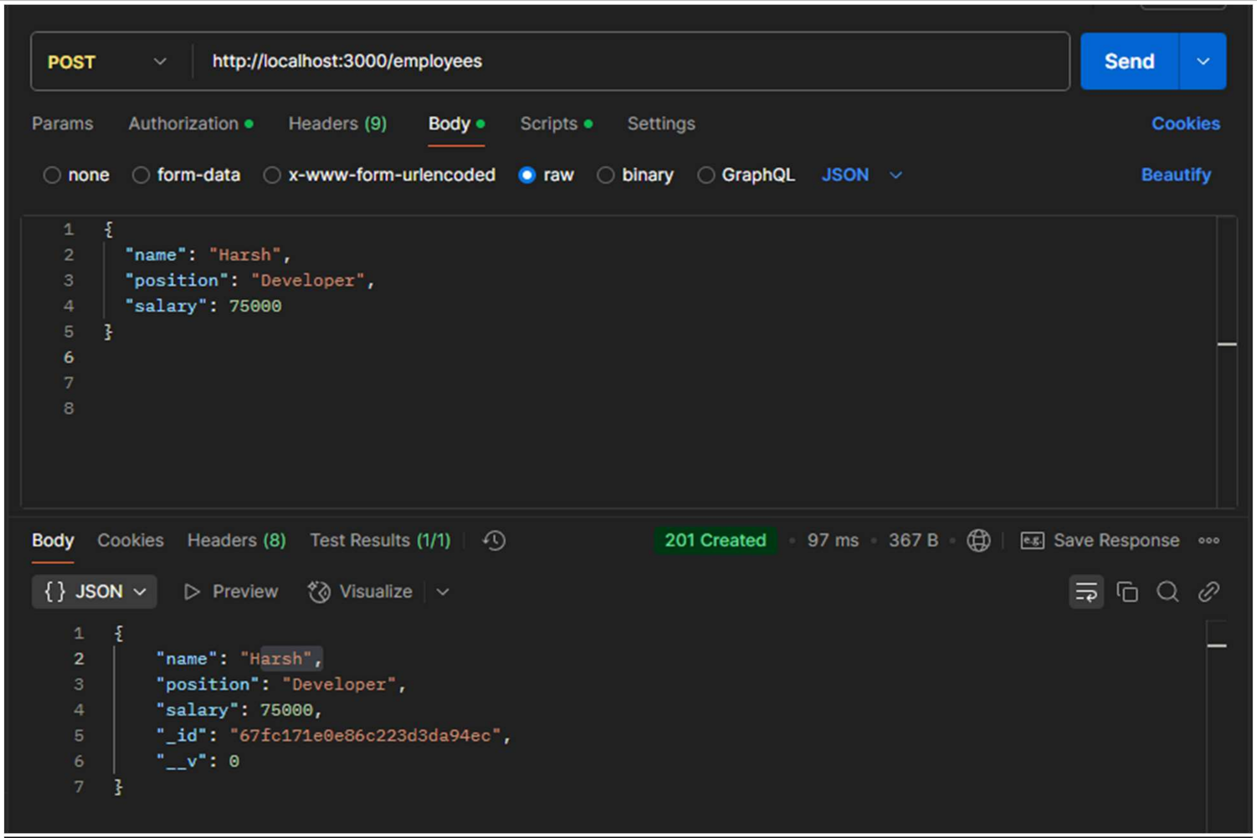
Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2N2ZjMTVlNDBlODZjMjIzZDNkYTk0ZTgiLCJ1c2VybmFtZSI6ImhhcnNoIiwiaWF0IjoxNzQ0NTc0MDExLCJleHAiOjE3NDQ1Nzc2MTF9.KxCrh2JSNkuKDgHALmX3pnnPcqh93dqVemn47wnXZuk

200 OK · 159 ms · 481 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
      eyJ1c2VySWQiOiI2N2ZjMTVlNDBlODZjMjIzZDNkYTk0ZTgiLCJ1c2VybmFtZSI6ImhhcnNoIiwiaWF0IjoxNzQ0NTc0MDk
      4LCJleHAiOjE3NDQ1Nzc2OTh9.OxUQ6LYW7iTjeFJkw2VNKSkpF-jvYgPZn3pBJkrhBWQ"
}
```

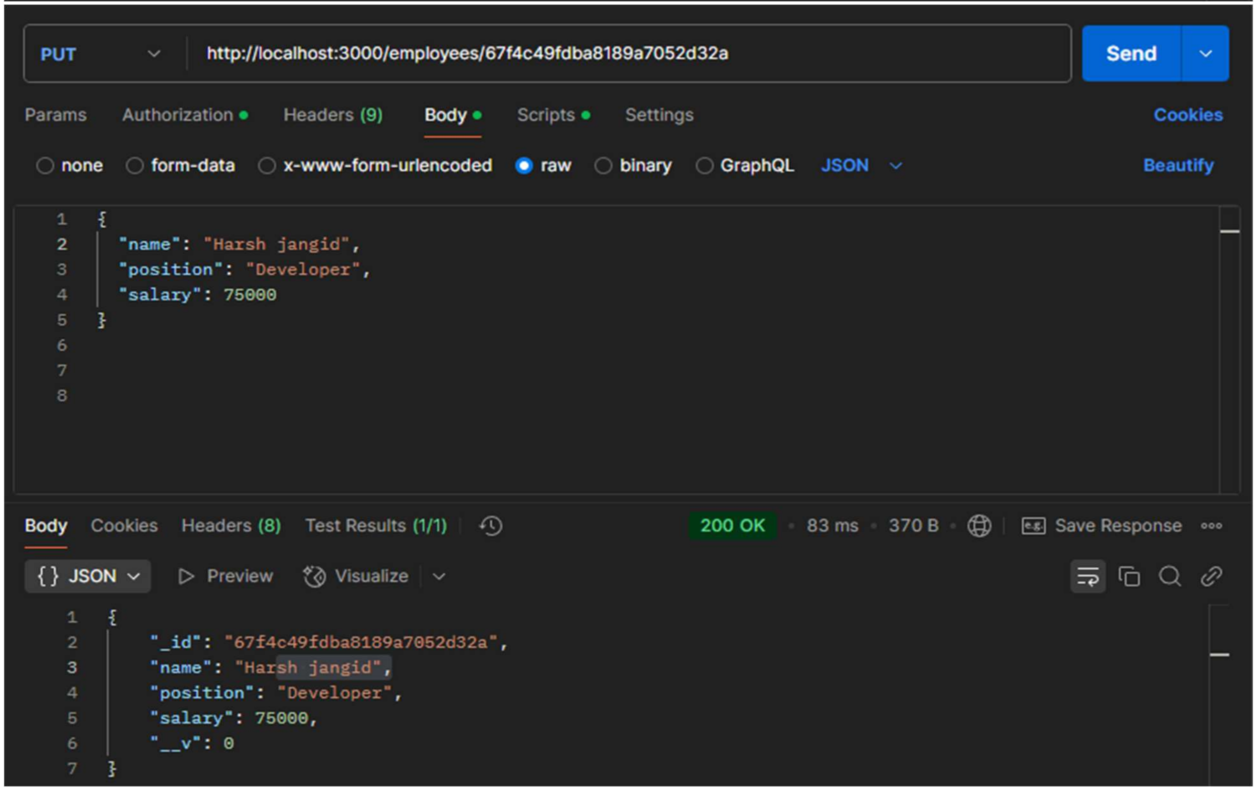


POST http://localhost:3000/employees

```
{
  "name": "Harsh",
  "position": "Developer",
  "salary": 75000
}
```

201 Created · 97 ms · 367 B

```
{
    "name": "Harsh",
    "position": "Developer",
    "salary": 75000,
    "_id": "67fc171e0e86c223d3da94ec",
    "__v": 0
}
```

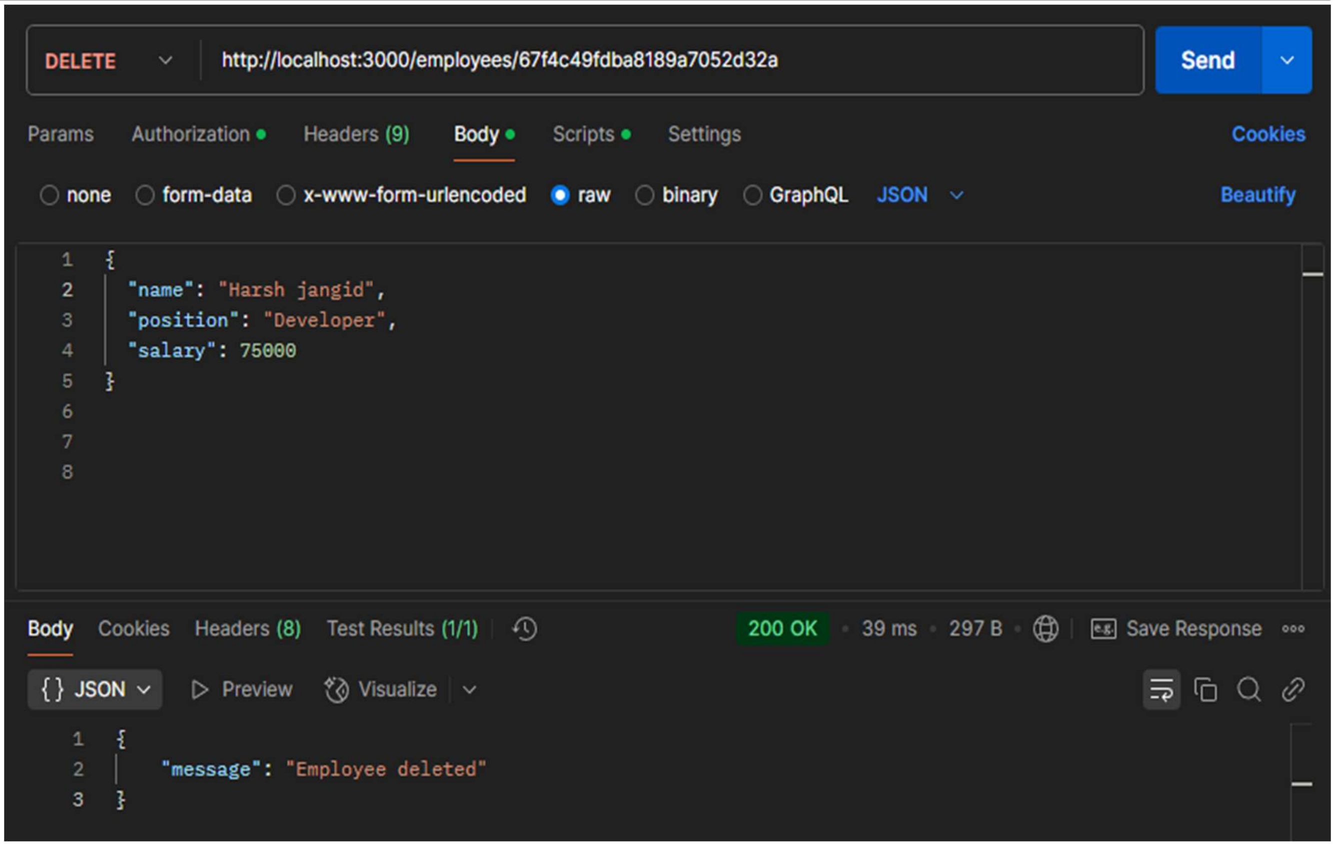


PUT http://localhost:3000/employees/67f4c49fdba8189a7052d32a

```
{
  "name": "Harsh jangid",
  "position": "Developer",
  "salary": 75000
}
```

200 OK · 83 ms · 370 B

```
{
    "_id": "67f4c49fdba8189a7052d32a",
    "name": "Harsh jangid",
    "position": "Developer",
    "salary": 75000,
    "__v": 0
}
```

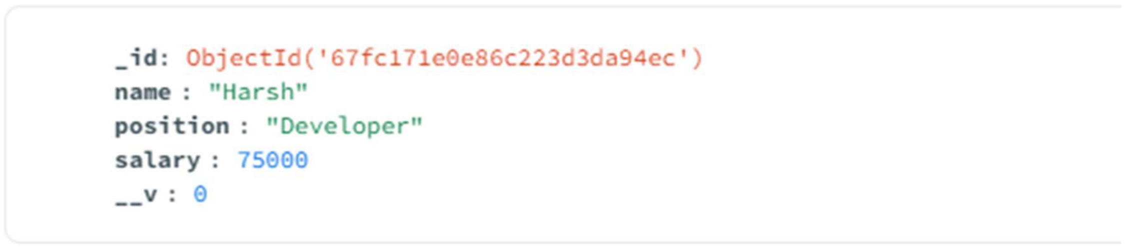


DELETE http://localhost:3000/employees/67f4c49fdba8189a7052d32a

```
{
  "name": "Harsh jangid",
  "position": "Developer",
  "salary": 75000
}
```

200 OK • 39 ms • 297 B

```
{
    "message": "Employee deleted"
}
```



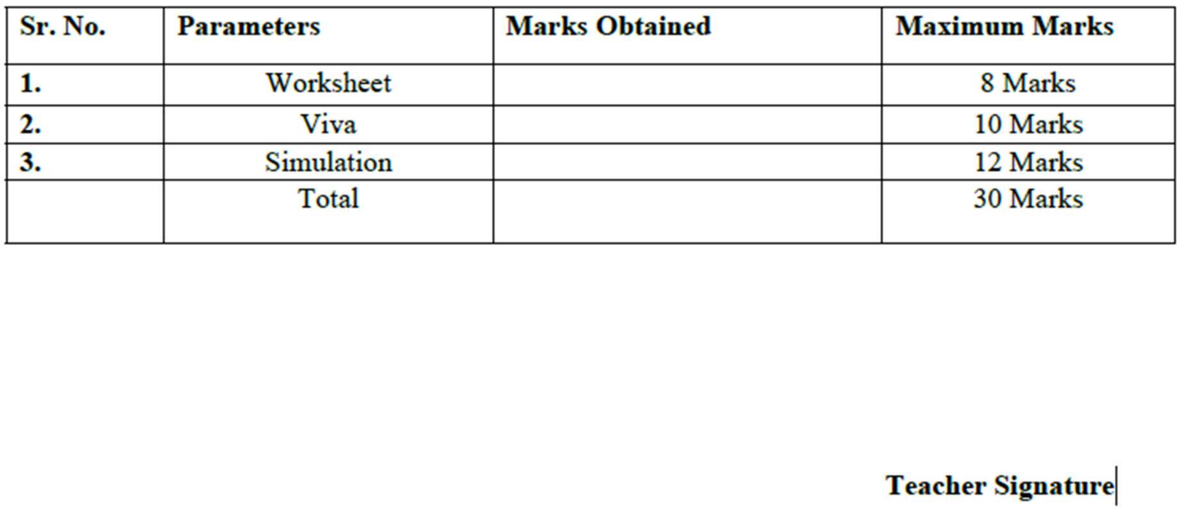
```
_id: ObjectId('67fc171e0e86c223d3da94ec')
name : "Harsh"
position : "Developer"
salary : 75000
__v : 0
```

5. <u>Learning outcomes (What I have learnt)</u>:

1. Understanding CRUD: Learn to perform Create, Read, Update, and Delete operations in Node.js with MongoDB.
2. Setup Node.js & MongoDB: Configure a Node.js server with Express and connect it to MongoDB using Mongoose.
3. Build RESTful APIs: Implement APIs for CRUD operations and test them using Postman.

4. Handle API Requests: Work with HTTP methods (GET, POST, PUT, DELETE) and manage request data.
5. Validate & Manage Data: Use Mongoose schemas for data validation and error handling in MongoDB.

Evalution Grid:



Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Worksheet		8 Marks
2.	Viva		10 Marks
3.	Simulation		12 Marks
	Total		30 Marks

Teacher Signature