MongoDB, MVC and Express CRUD App

Instructor: Pramod Kumar Jena

---

1. Query and Params in Express.js

Query Parameters

- **Definition**: Query parameters are key-value pairs appended to the URL after a `?`. Used to pass optional data to the server.

Syntax:

```
http://example.com/api/users?name=John&age=25
```

Example in Express:

```
app.get('/api/users', (req, res) => {
  const { name, age } = req.query;
  res.send(`User: ${name}, Age: ${age}`);
});
```

Route Parameters

- **Definition**: Route parameters are used to capture specific values from the URL path.

Syntax:

```
http://example.com/api/users/:id
```

Example in Express:

```
app.get('/api/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User ID: ${userId}`);
```

```
});
```

---

2. MongoDB and Mongoose

MongoDB

- A NoSQL database for storing data in JSON-like documents.
- **Core Commands**:
  - `show dbs`: List all databases.
  - `use <dbname>`: Switch to a database.
  - `db.createCollection('collectionName')`: Create a new collection.
  - `db.collectionName.insertOne({})`: Insert a document.
  - `db.collectionName.find()`: Query documents.

Mongoose

- An Object Data Modeling (ODM) library for MongoDB in Node.js.
- Simplifies interactions with MongoDB.

---

3. Schema and Model

Schema

- **Definition**: A blueprint for the structure of documents in a MongoDB collection.

Example:

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: String,
  age: Number,
  email: String
});
```

Model

- **Definition**: A wrapper for the schema, used for creating and interacting with documents.

Example:

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

---

4. MVC Architecture

- **Model**: Represents the data (e.g., Mongoose models).
- **View**: Handles the UI.
- **Controller**: Contains the logic for handling requests and responses.

---

5. MongoDB and Mongoose Cheat Sheet

Common MongoDB Commands

Command	Description
`db.collection.find()`	Retrieve all documents.
`db.collection.find({ key: value })`	Find documents with conditions.
`db.collection.updateOne()`	Update a specific document.
`db.collection.deleteOne()`	Delete a specific document.

Common Mongoose Methods

Method	Description
`Model.find()`	Retrieve all documents.
`Model.findById()`	Find a document by ID.
`Model.create()`	Insert a new document.

`Model.updateOne()`	Update a specific document.
`Model.deleteOne()`	Delete a specific document.

---

6. Express CRUD App with MongoDB

Setup
Initialize a Node.js project:

```
npm init -y
npm install express mongoose body-parser
```

Create the file structure:

```
/project
├── /models
│   └── user.js
├── /controllers
│   └── userController.js
├── /routes
│   └── userRoutes.js
└── server.js
```

Model (user.js)

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

const userSchema = new mongoose.Schema({
  name: String,
  age: Number,
  email: { type: String, required: true, unique: true }
});

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

Controller (userController.js)

```
const User = require('../models/user');

exports.getAllUsers = async (req, res) => {
  const users = await User.find();
  res.json(users);
};

exports.createUser = async (req, res) => {
  const newUser = new User(req.body);
  await newUser.save();
  res.status(201).json(newUser);
};

exports.updateUser = async (req, res) => {
  const { id } = req.params;
  const updatedUser = await User.findByIdAndUpdate(id, req.body, {
new: true });
  res.json(updatedUser);
};

exports.deleteUser = async (req, res) => {
  const { id } = req.params;
  await User.findByIdAndDelete(id);
  res.status(204).send();
};
```

Routes (userRoutes.js)

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.get('/', userController.getAllUsers);
router.post('/', userController.createUser);
router.put('/:id', userController.updateUser);
```

```
router.delete('/:id', userController.deleteUser);

module.exports = router;
```

Server (server.js)

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const userRoutes = require('./routes/userRoutes');

const app = express();
app.use(bodyParser.json());
app.use('/api/users', userRoutes);

mongoose.connect('mongodb://localhost:27017/mydb', { useNewUrlParser:
true, useUnifiedTopology: true })
  .then(() => {
    console.log('Connected to MongoDB');
    app.listen(3000, () => console.log('Server running on port
3000'));
  })
  .catch(err => console.error(err));
```

---

7. Task for Students

- Add a new endpoint for fetching a user by their email.
- Create a MongoDB collection using the CLI and insert sample data to test your API.

---

8. Resources

- MongoDB Official Documentation
- Mongoose Documentation