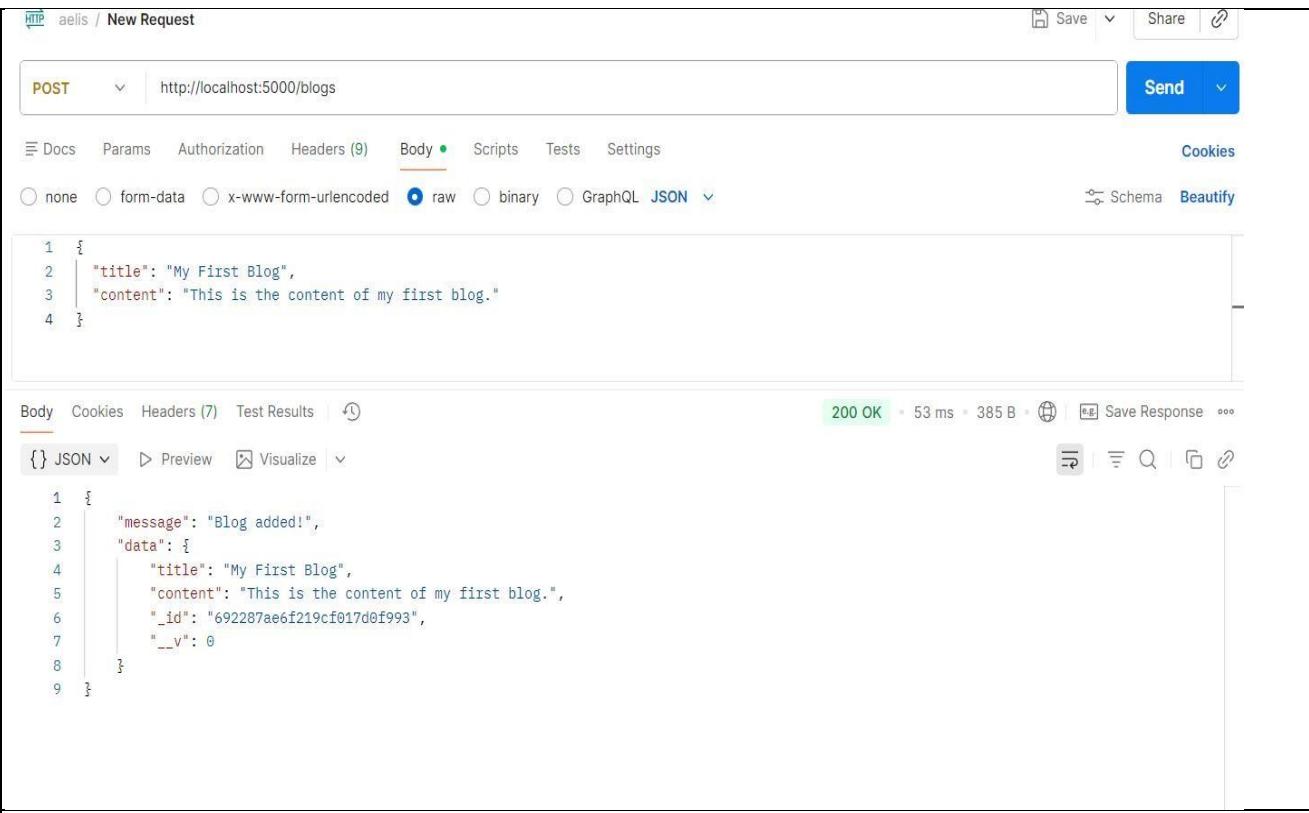


1	<p><b>While building a blog application, users should be able to add new blog posts to the database.</b></p> <p><b>Sometimes, they might forget to include the title. Create a Mongoose model with validation to ensure that each blog post has both title and content.</b></p> <p><b>Additionally, configure your application to handle invalid submissions by returning a proper JSON error response.</b></p>
Code	<pre>// server.js const express = require("express"); const mongoose = require("mongoose"); const blogRoutes = require("./routes/blogRoutes"); const app = express(); app.use(express.json()); mongoose   .connect("mongodb://127.0.0.1:27017/practical5")   .then(() =&gt; console.log("✓ Connected to MongoDB"))   .catch((err) =&gt; console.log("✗ MongoDB Error:", err.message)); app.use("/blogs", blogRoutes); const PORT = 5000; app.listen(PORT, () =&gt; console.log(`Q1 Server running at http://localhost:\${PORT}`));  // models/Blog.js const mongoose = require("mongoose"); const blogSchema = new mongoose.Schema({   title: { type: String, required: [true, "Title is required"] },   content: { type: String, required: [true, "Content is required"] }, }); module.exports = mongoose.model("Blog", blogSchema); // controllers/blogController.js const Blog = require("../models/Blog"); exports.createBlog = async (req, res) =&gt; {   try {     const post = await Blog.create(req.body);     res.json({ message: "Blog added!", data: post });   } catch (err) {     res.status(400).json({ error: err.message });   } }; // routes/blogRoutes.js const express = require("express"); const { createBlog } = require("../controllers/blogController"); const router = express.Router(); router.post("/", createBlog); module.exports = router;</pre>

<b>outpu</b> <b>t</b>	 <pre> 1 { 2   "title": "My First Blog", 3   "content": "This is the content of my first blog." 4 }       </pre> <p>Body Cookies Headers (7) Test Results ⚡</p> <p>200 OK • 53 ms • 385 B • Save Response ⚡</p> <p>{ } JSON ▾ Preview Visualize</p> <pre> 1 { 2   "message": "Blog added!", 3   "data": { 4     "title": "My First Blog", 5     "content": "This is the content of my first blog.", 6     "_id": "692287ae6f219cf017d0f993", 7     "__v": 0 8   } 9 }       </pre>
<b>2</b>	<b>In a task management system, implement a Mongoose query to fetch and display all tasks where the status field is set to "completed"</b>
<b>Code</b>	<pre> // server.js const express = require("express"); const mongoose = require("mongoose"); const taskRoutes = require("./routes/taskRoutes"); const app = express(); app.use(express.json()); mongoose .connect("mongodb://127.0.0.1:27017/practical5") .then(() =&gt; console.log("✓ Q2 MongoDB connected")) .catch((err) =&gt; console.log("✗", err.message)); app.use("/", taskRoutes); const PORT = 5002; app.listen(PORT, () =&gt; console.log(`Q2 Server running at http://localhost:\${PORT}`)); // models/Task.js const mongoose = require("mongoose"); const taskSchema = new mongoose.Schema({ task: String, status: String, }); module.exports = mongoose.model("Task", taskSchema); // controllers/taskController.js const Task = require("../models/Task"); exports.getCompletedTasks = async (req, res) =&gt; { try { const tasks = await Task.find({ status: "completed" }); res.json(tasks); } catch (error) { res.status(500).json({ message: "Internal Server Error" }); } }       </pre>

```

} catch (err) {
res.status(500).json({ error: err.message });
}
};

// routes/taskRoutes.js
const express = require("express");
const { getCompletedTasks } = require("../controllers/taskController");
const router = express.Router();
router.get("/tasks/completed", getCompletedTasks);
module.exports = router;

```

**Output**

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** http://localhost:5000/tasks/completed
- Headers:** Headers (9) - raw
- Body:** Raw JSON response (copied below)
- Test Results:** 200 OK, 12 ms, 329 E

```

5   },
6   {
7     "task": "Wash dishes",
8     "status": "pending"
9   }
10 ]

```

```

1 [
2   {
3     "_id": "692288b87ee7ffbeb276dee2",
4     "task": "Complete assignment",
5     "status": "completed",
6     "__v": 0
7   }
8 ]

```

**3** You are working on a Task Management System where users can create tasks with different statuses, such as "completed", "in-progress", and "pending". You need to implement a Mongoose query to fetch and display all tasks that have the status field set to "completed".

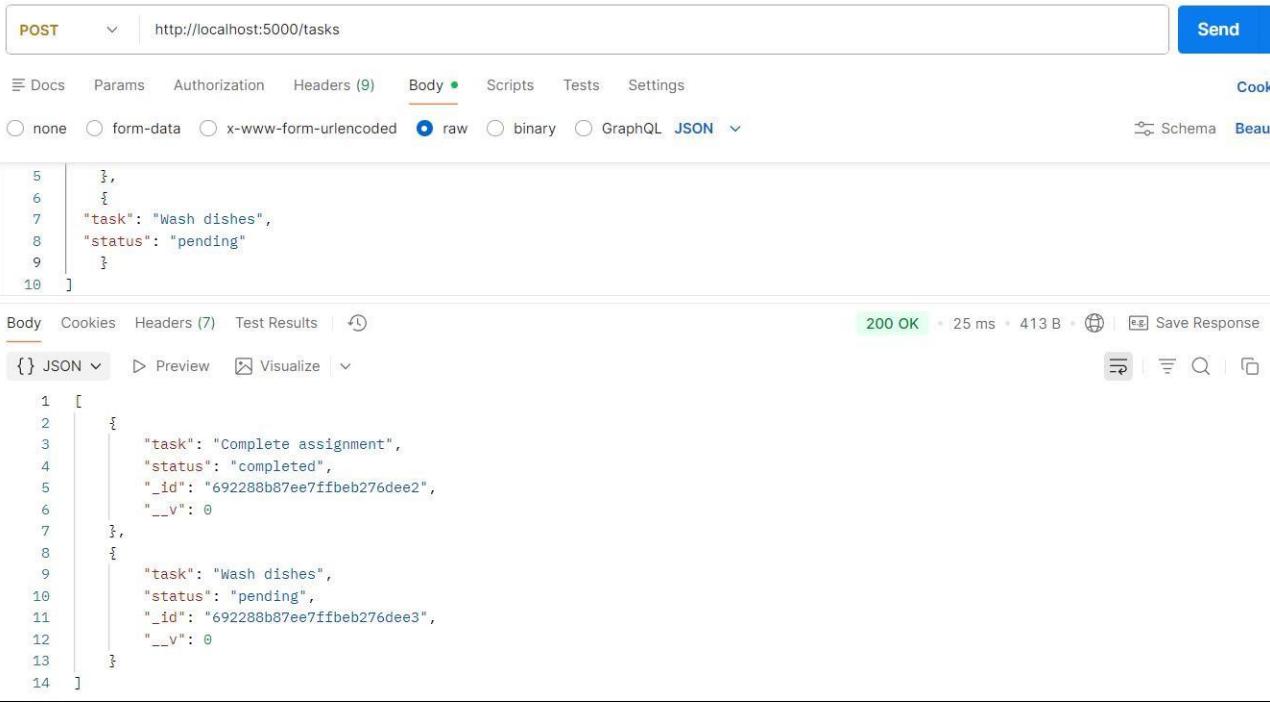
**Code**

```

// server.js
const express = require("express");
const mongoose = require("mongoose");
const taskRoutes = require("./routes/taskRoutes");
const app = express();
app.use(express.json());
mongoose
.connect("mongodb://127.0.0.1:27017/practical5")
.then(() => console.log("✓ Q3 MongoDB connected"))
.catch((err) => console.log("✗", err.message));
app.use("/", taskRoutes);
const PORT = 5003;

```

```
app.listen(PORT, () => console.log(`Q3 Server running at http://localhost:${PORT}`));
// models/Task.js
const mongoose = require("mongoose");
const taskSchema = new mongoose.Schema({
task: String,
status: String,
});
module.exports = mongoose.model("Task", taskSchema);
// controllers/taskController.js
const Task = require("../models/Task");
exports.createTask = async (req, res) => {
try {
const task = await Task.create(req.body);
res.json(task);
} catch (err) {
res.status(400).json({ error: err.message });
}
};
exports.getCompletedTasks = async (req, res) => {
try {
const tasks = await Task.find({ status: "completed" });
res.json(tasks);
} catch (err) {
res.status(500).json({ error: err.message });
}
};
// routes/taskRoutes.js
const express = require("express");
const { createTask, getCompletedTasks } = require("../controllers/taskController");
const router = express.Router();
router.post("/tasks", createTask);
router.get("/tasks/completed", getCompletedTasks);
module.exports = router;
```

	 <p>The screenshot shows a Postman interface with a POST request to <code>http://localhost:5000/tasks</code>. The request body is raw JSON:</p> <pre> 5   }, 6   { 7     "task": "Wash dishes", 8     "status": "pending" 9   } 10 ] </pre> <p>The response is a 200 OK status with a timestamp of 25 ms, a size of 413 B, and a global icon. The response body is also raw JSON:</p> <pre> 1 [ 2   { 3     "task": "Complete assignment", 4     "status": "completed", 5     "_id": "692288b87ee7ffbeb276dee2", 6     "__v": 0 7   }, 8   { 9     "task": "Wash dishes", 10    "status": "pending", 11    "_id": "692288b87ee7ffbeb276dee3", 12    "__v": 0 13  } 14 ] </pre>
4	<p><b>You are building an admin panel where admins can delete multiple inactive users from the database in one go. Write the Mongoose query to remove all documents with isActive: false and send a success response with the number of deleted documents.</b></p>
Code	<pre>// models/User.js const mongoose = require("mongoose"); const userSchema = new mongoose.Schema({   name: String,   email: String,   isActive: { type: Boolean, default: true }, }); module.exports = mongoose.model("User", userSchema); // controllers/userController.js const User = require("../models/User"); exports.createUser = async (req, res) =&gt; {   try {     const u = await User.create(req.body);     res.json(u);   } catch (err) {     res.status(400).json({ error: err.message });   } }; exports.deleteInactiveUsers = async (req, res) =&gt; {   try {     const result = await User.deleteMany({ isActive: false });     res.json({ message: "Inactive users deleted", deletedCount: result.deletedCount });   } catch (err) {     res.status(500).json({ error: err.message });   } };</pre>

```
}

};

// routes/userRoutes.js
const express = require("express");
const { createUser, deleteInactiveUsers } = require("../controllers/userController");
const router = express.Router();
router.post("/users", createUser);
router.delete("/users/inactive", deleteInactiveUsers);
module.exports = router;
// server.js
const express = require("express");
const mongoose = require("mongoose");
const userRoutes = require("./routes/userRoutes");
const app = express();
app.use(express.json());
mongoose.connect("mongodb://127.0.0.1:27017/practical5")
.then(()=>console.log("✓Q4 MongoDB connected"))
.catch(err=>console.log("✗", err.message));
app.use("/", userRoutes);
app.listen(5004, ()=>console.log("Q4 Server running on http://localhost:5004"));
```

## Output

**POST** ▾ | http://localhost:5000/users

Docs Params Authorization Headers (9) Body • Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON** ▾

```

5   "isActive": true
6 },
7 {
8   "name": "Inactive User",
9   "email": "inactive@example.com",
10  "isActive": false
11 }

```

Body Cookies Headers (7) Test Results | ⚡ 200 OK • 121 ms •

{ } JSON ▾ ▷ Preview Visualize ▾

```

1 [
2 {
3   "name": "Active User",
4   "email": "active@example.com",
5   "isActive": true,
6   "_id": "69228935ede4395229a8b470",
7   "__v": 0
8 },
9 {
10  "name": "Inactive User",
11  "email": "inactive@example.com",
12  "isActive": false,
13  "_id": "69228935ede4395229a8b471",
14  "__v": 0
15 }
16 ]

```

---

**DELETE** ▾ | http://localhost:5000/users/inactive

Docs Params Authorization Headers (9) Body • Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON** ▾

```

5   "isActive": true
6 },
7 {
8   "name": "Inactive User",
9   "email": "inactive@example.com",
10  "isActive": false
11 }

```

Body Cookies Headers (7) Test Results | ⚡ 200 OK • 121 ms •

{ } JSON ▾ ▷ Preview Visualize ▾

```

1 {
2   "message": "Inactive users deleted",
3   "deletedCount": 1
4 }

```

5 While designing a blogging platform, each blog post should store its comments.

	<b>Create a Mongoose schema that embeds comments inside the post document. Each comment should have author and message</b>
<b>Code</b>	<pre>// models/Post.js const mongoose = require("mongoose"); const commentSchema = new mongoose.Schema({   author: String,   message: String,   createdAt: { type: Date, default: Date.now }, }); const postSchema = new mongoose.Schema({   title: String,   content: String,   comments: [commentSchema],   createdAt: { type: Date, default: Date.now }, }); module.exports = mongoose.model("Post", postSchema); // controllers/postController.js const Post = require("../models/Post"); exports.createPost = async (req, res) =&gt; {   try {     const p = await Post.create(req.body);     res.json(p);   } catch (err) {     res.status(400).json({ error: err.message });   } }; exports.addComment = async (req, res) =&gt; {   try {     const post = await Post.findById(req.params.id);     post.comments.push(req.body);     await post.save();     res.json(post);   } catch (err) {     res.status(400).json({ error: err.message });   } }; exports.getPosts = async (req, res) =&gt; {   const posts = await Post.find();   res.json(posts); }; // routes/postRoutes.js const express = require("express"); const { createPost, addComment, getPosts } = require("../controllers/postController"); const router = express.Router(); router.post("/posts", createPost); router.get("/posts", getPosts); router.post("/posts/:id/comments", addComment); module.exports = router;</pre>

```
// server.js
const express = require("express");
const mongoose = require("mongoose");
const postRoutes = require("./routes/postRoutes");
const app = express();
app.use(express.json());
mongoose.connect("mongodb://127.0.0.1:27017/practical5")
.then(()=>console.log("✓ Q5 MongoDB connected"))
.catch(err=>console.log("✗", err.message));
app.use("/", postRoutes);
app.listen(5005, ()=>console.log("Q5 Server running on http://localhost:5005"));
```

**Output**

POST http://localhost:5000/posts-with-comments

Body (raw) JSON

```

1 {
2   "title": "Post with comments",
3   "content": "This is a blog post.",
4   "comments": [
5     { "author": "Divya", "message": "Nice article!" },
6     { "author": "Javerchanfd", "message": "Very helpful!" }
7   ]
8 }
```

200 OK 73 ms 516 B Save Response

```

1 {
2   "title": "Post with comments",
3   "content": "This is a blog post.",
4   "comments": [
5     {
6       "author": "Divya",
7       "message": "Nice article!",
8       "_id": "6922aa3a3a0e3560a9036626"
9     },
10    {
11      "author": "Javerchanfd",
12      "message": "Very helpful!",
13      "_id": "6922aa3a3a0e3560a9036627"
14    }
15  ],
16  "_id": "6922aa3a3a0e3560a9036625",
17  "__v": 0
18 }
```

**6** In a student management application, each student can enroll in multiple courses.

Design a normalized schema where student documents reference course documents.

<b>Code</b>	<pre>// models/Course.js const mongoose = require("mongoose"); const courseSchema = new mongoose.Schema({ courseName: String }); module.exports = mongoose.model("Course", courseSchema); // models/Student.js const mongoose = require("mongoose"); const studentSchema = new mongoose.Schema({ studentName: String, courses: [{ type: mongoose.Schema.Types.ObjectId, ref: "Course" }], }); module.exports = mongoose.model("Student", studentSchema); // controllers/courseController.js const Course = require("../models/Course"); exports.createCourse = async (req, res) =&gt; { try { const c = await Course.create(req.body); res.json(c); } catch(err){ res.status(400).json({ error: err.message }); } }; exports.getCourses = async (req,res)=&gt;{ res.json(await Course.find()); }; // controllers/studentController.js const Student = require("../models/Student"); exports.createStudent = async (req, res) =&gt; { try { const s = await Student.create(req.body); res.json(s); } catch(err){ res.status(400).json({ error: err.message }); } }; exports.getStudents = async (req,res)=&gt; { const students = await Student.find().populate("courses"); res.json(students); }; // routes/courseRoutes.js const express = require("express"); const { createCourse, getCourses } = require("../controllers/courseController"); const router = express.Router(); router.post("/courses", createCourse); router.get("/courses", getCourses); module.exports = router; // routes/studentRoutes.js const express = require("express"); const { createStudent, getStudents } = require("../controllers/studentController"); const router = express.Router(); router.post("/students", createStudent); router.get("/students", getStudents); module.exports = router; // server.js const express = require("express"); const mongoose = require("mongoose"); const courseRoutes = require("./routes/courseRoutes"); const studentRoutes = require("./routes/studentRoutes"); const app = express(); app.use(express.json()); mongoose.connect("mongodb://127.0.0.1:27017/practical5")</pre>
-------------	---

```
.then(()=>console.log("✓Q6 MongoDB connected"))
.catch(err=>console.log("✗, err.message));
app.use("/", courseRoutes);
app.use("/", studentRoutes);
app.listen(5006, ()=>console.log("Q6 Server running on http://localhost:5006"));
```

**Outp  
ut**

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:5000/courses
- Body:** JSON (raw JSON input: {"courseName": "BSc IT - MongoDB"} and raw response JSON output: {"courseName": "BSc IT - MongoDB", "\_id": "69228a3d0a3459140f30313f", "\_\_v": 0})
- Headers:** (9 items)
- Response:** 200 OK (132 ms, 309 B, JSON)

The screenshot shows a Postman interface with a POST request to `http://localhost:5000/students`. The request body is a JSON object:

```

1  {
2    "studentName": "Divya",
3    "courses": ["6922aa983a0e3560a9036629"]
4  }
5

```

The response is a 200 OK status with the following JSON data:

```

1  {
2    "studentName": "Divya",
3    "courses": [
4      "6922aa983a0e3560a9036629"
5    ],
6    "_id": "6922aac93a0e3560a903662b",
7    "__v": 0
8  }

```

7	<p><b>You're building a social media app where users can like posts and interact with each other's profiles.</b></p> <p><b>Your task is to design the database schema using Mongoose (a MongoDB ODM for Node.js) with a hybrid data model, where:</b></p> <ol style="list-style-type: none"> <li><b>1. Likes are stored directly in the post document (embedded data).</b></li> <li><b>2. User profiles are stored as references in the database, linking users to the posts they like (referenced data).</b></li> </ol>
---	--

```

code // models/HybridUser.js
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema({ username: String });
module.exports = mongoose.model("HybridUser", userSchema);
// models/HybridPost.js
const mongoose = require("mongoose");
const likeSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: "HybridUser" },
  likedAt: { type: Date, default: Date.now }
});
const postSchema = new mongoose.Schema({
  text: String,
  likes: [likeSchema],
  createdAt: { type: Date, default: Date.now }
});
module.exports = mongoose.model("HybridPost", postSchema);
// controllers/hybridController.js
const HybridUser = require("../models/HybridUser");
const HybridPost = require("../models/HybridPost");
exports.createUser = async (req,res)=>{
  const u = await HybridUser.create(req.body);
  res.json(u);
}

```

```

};

exports.createPost = async (req,res)=>{
const p = await HybridPost.create(req.body);
res.json(p);
};

exports.likePost = async (req,res)=>{
const post = await HybridPost.findById(req.params.id);
post.likes.push({ user: req.body.user });
await post.save();
res.json(post);
};

exports.getPosts = async (req,res)=>{
const posts = await HybridPost.find().populate("likes.user", "username");
res.json(posts);
};

// routes/hybridRoutes.js
const express = require("express");
const { createUser, createPost, likePost, getPosts } = require("../controllers/hybridController");
const router = express.Router();
router.post("/hybrid-users", createUser);
router.post("/hybrid-posts", createPost);
router.post("/hybrid-posts/:id/like", likePost);
router.get("/hybrid-posts", getPosts);
module.exports = router;
// server.js
const express = require("express");
const mongoose = require("mongoose");
const hybridRoutes = require("./routes/hybridRoutes");
const app = express();
app.use(express.json());
mongoose.connect("mongodb://127.0.0.1:27017/practical5")
.then(()=>console.log("✓Q7 MongoDB connected"))
.catch(err=>console.log("✗", err.message));
app.use("/", hybridRoutes);
app.listen(5007, ()=>console.log("Q7 Server running on http://localhost:5007"));

```

## Outp ut

**POST** ▾ http://localhost:5000/hybrid-users

Docs Params Authorization Headers (9) Body • Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON** ▾

```

1  {
2    "username": "user1"
3  }
4

```

Body Cookies Headers (7) Test Results | ⏱

200 OK • 87 ms • 296 B • (

{ } JSON ▾ ▶ Preview ⚡ Visualize ▾

```

1  {
2    "username": "user1",
3    "_id": "69228ab48cc52738ce18a523",
4    "__v": 0
5

```

**POST** ▾ http://localhost:5000/hybrid-posts

Docs Params Authorization Headers (9) Body • Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON** ▾

Sc

```

2  {
3    "text": "Hello social world!",
4    "likes": [
5      {
6        "user": "69228ab48cc52738ce18a523"
7      }
8    ]
9  }
10
11

```

Body Cookies Headers (7) Test Results | ⏱

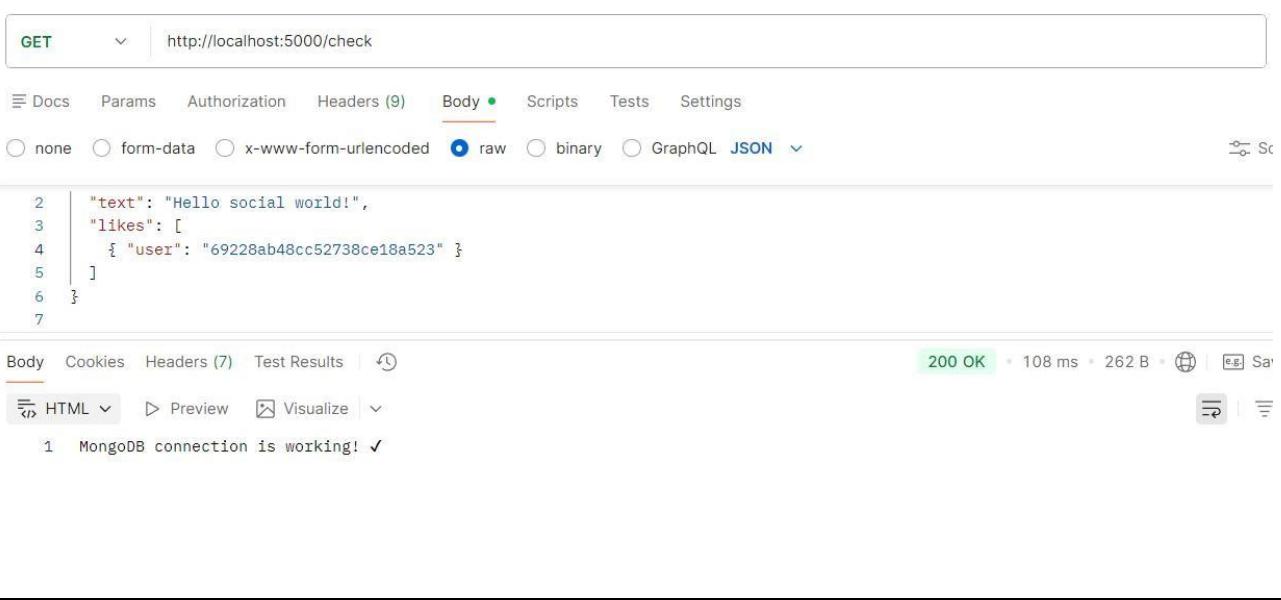
200 OK • 27 ms • 386 B • 🌐 Sav

{ } JSON ▾ ▶ Preview ⚡ Visualize ▾

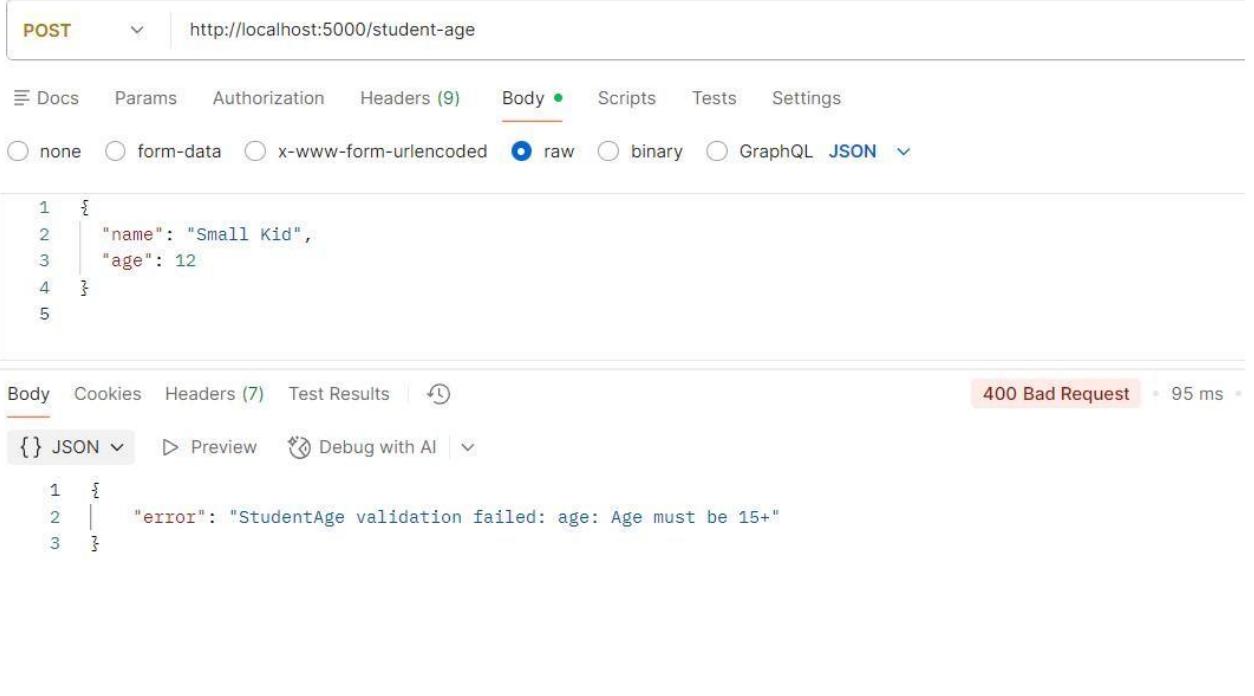
```

1  {
2    "text": "Hello social world!",
3    "likes": [
4      {
5        "user": "69228ab48cc52738ce18a523",
6        "_id": "69228ae78cc52738ce18a526"
7      }
8    ],
9    "_id": "69228ae78cc52738ce18a525",
10   "__v": 0
11

```

8	<p><b>While developing a Node.js application, connect it to a local MongoDB instance using Mongoose. Write the code to establish the connection and log "Connected to MongoDB" on success, or an error message otherwise.</b></p>
Code	<pre>// server.js const express = require("express"); const mongoose = require("mongoose"); const app = express(); mongoose.connect("mongodb://127.0.0.1:27017/practical5") .then(()=&gt;console.log("✓ Q8 MongoDB connected")) .catch(err=&gt;console.log("✗", err.message)); app.get("/check", (req,res)=&gt;res.send("MongoDB connection is working! ✓")); app.listen(5008, ()=&gt;console.log("Q8 Server running on http://localhost:5008"));</pre>
Output	 <p>The screenshot shows a Postman request to <code>http://localhost:5008/check</code>. The response body is a JSON object:</p> <pre> 2   "text": "Hello social world!", 3   "likes": [ 4     { "user": "69228ab48cc52738ce18a523" } 5   ] 6 7 </pre> <p>The status bar at the bottom indicates a <b>200 OK</b> response with <b>108 ms</b> latency and <b>262 B</b> size.</p>
9	<p><b>While developing a school app, ensure that students younger than 15 cannot be saved to the database. Implement this using Mongoose schema validation.</b></p>

<b>Code</b>	<pre>// models/StudentAge.js const mongoose = require("mongoose"); const studentAgeSchema = new mongoose.Schema({   name: String,   age: { type: Number, min: [15, "Age must be 15+"] } }); module.exports = mongoose.model("StudentAge", studentAgeSchema); // controllers/studentAgeController.js const StudentAge = require("../models/StudentAge"); exports.create = async (req,res) =&gt; {   try {     const s = await StudentAge.create(req.body);     res.json(s);   } catch (err) {     res.status(400).json({ error: err.message });   } }; exports.getAll = async (req,res) =&gt; res.json(await StudentAge.find()); // routes/studentAgeRoutes.js const express = require("express"); const { create, getAll } = require("../controllers/studentAgeController"); const router = express.Router(); router.post("/student-age", create); router.get("/student-age", getAll); module.exports = router; // server.js const express = require("express"); const mongoose = require("mongoose"); const routes = require("./routes/studentAgeRoutes"); const app = express(); app.use(express.json()); mongoose.connect("mongodb://127.0.0.1:27017/practical5")   .then(()=&gt;console.log("✓Q9 MongoDB connected"))   .catch(err=&gt;console.log("✗", err.message)); app.use("/", routes); app.listen(5009, ()=&gt;console.log("Q9 Server running on http://localhost:5009")); </pre>
-------------	---

Output	 <pre> POST http://localhost:5000/student-age  Body (raw JSON) {   "name": "Small Kid",   "age": 12 }  400 Bad Request • 95 ms [{"error": "StudentAge validation failed: age: Age must be 15+"} ] </pre>
10	<p>You are building a user registration API for a web application. When a new user tries to register, you need to ensure that the email address they provide is unique.</p> <p>Implement the registration logic so that if a user attempts to register with an email that already exists in the database, the API should respond with a JSON error.</p>
Code	<pre> // models/RegUser.js const mongoose = require("mongoose"); const regUserSchema = new mongoose.Schema({   name: String,   email: { type: String, unique: true, required: true } }); module.exports = mongoose.model("RegUser", regUserSchema); // controllers/regController.js const RegUser = require("../models/RegUser"); exports.register = async (req, res) =&gt; {   try {     const u = await RegUser.create(req.body);     res.json({ message: "User registered", data: u });   } catch (err) {     if (err.code === 11000) return res.status(400).json({ error: "Email already exists" });     res.status(500).json({ error: err.message });   } }; exports.getUsers = async (req, res) =&gt; res.json(await RegUser.find()); // routes/regRoutes.js const express = require("express"); const { register, getUsers } = require("../controllers/regController"); const router = express.Router(); router.post("/register", register); </pre>

```

router.get("/register", getUsers);
module.exports = router;
// server.js
const express = require("express");
const mongoose = require("mongoose");
const regRoutes = require("./routes/regRoutes");
const app = express();
app.use(express.json());
mongoose.connect("mongodb://127.0.0.1:27017/practical5")
.then(()=>console.log("✓Q10 MongoDB connected"))
.catch(err=>console.log("✗, err.message));
app.use("/", regRoutes);

app.listen(5010, ()=>console.log("Q10 Server running on http://localhost:5010"));

```

**O/p**

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:5000/register
- Headers:** (9 items shown)
- Body:** (raw JSON)

```

1  {
2   "name": "Test User",
3   "email": "test@example.com"
4 }
5

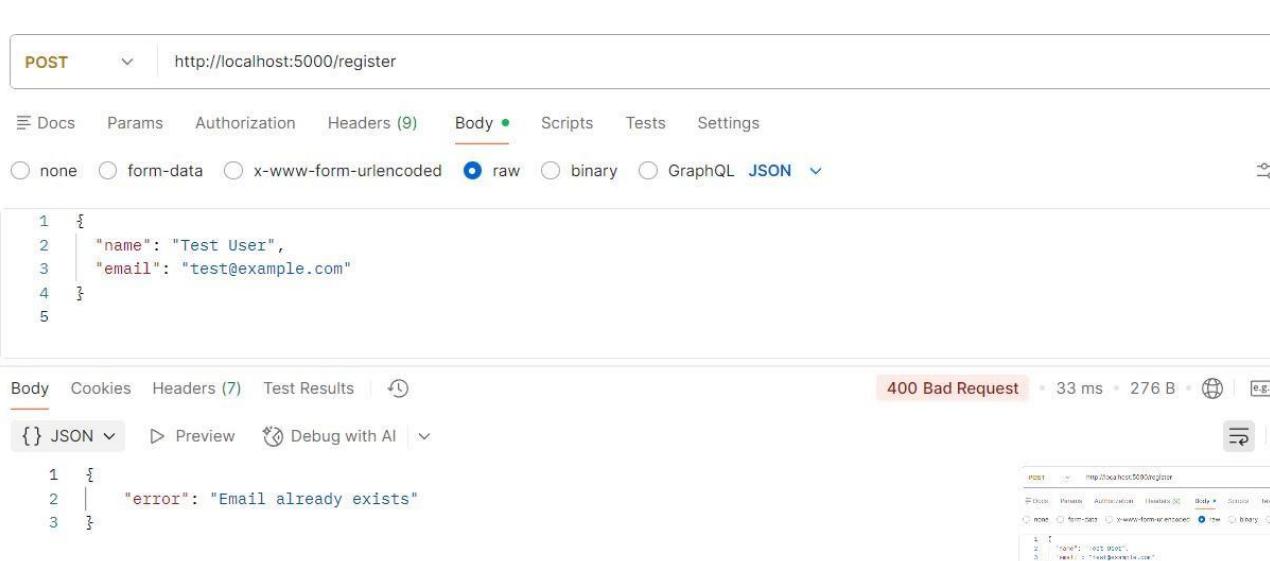
```

- Response:** 200 OK (109 ms, 361 B)

```

{
  "message": "User registered",
  "data": {
    "name": "Test User",
    "email": "test@example.com",
    "_id": "69228b95327f835d584155ad",
    "__v": 0
  }
}

```

	
11	<p><b>You are building a simple user management system using Express.js and MongoDB. Perform the following tasks:</b></p> <ul style="list-style-type: none"> <li>• Connect to MongoDB using Mongoose in your Express.js application.</li> <li>• Create a User model with fields: name, email, and password.</li> </ul>
Code	<pre>// models/User.js const mongoose = require("mongoose"); const userSchema = new mongoose.Schema({   name: String,   email: String,   isActive: { type: Boolean, default: true } }); module.exports = mongoose.model("User", userSchema); // controllers/userController.js const User = require("../models/User"); exports.create = async (req, res) =&gt; res.json(await User.create(req.body)); exports.getAll = async (req, res) =&gt; res.json(await User.find()); exports.getById = async (req, res) =&gt; res.json(await User.findById(req.params.id)); exports.update = async (req, res) =&gt; {   const u = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });   res.json(u); }; exports.remove = async (req, res) =&gt; {   await User.findByIdAndDelete(req.params.id);   res.json({ message: "User deleted" }); }; // routes/userRoutes.js const express = require("express"); const { create, getAll, getById, update, remove } = require("../controllers/userController"); const router = express.Router(); router.post("/api/users", create); router.get("/api/users", getAll);</pre>

```

router.get("/api/users/:id", getById);
router.put("/api/users/:id", update);
router.delete("/api/users/:id", remove);
module.exports = router;
// server.js
const express = require("express");
const mongoose = require("mongoose");
const userRoutes = require("./routes/userRoutes");
const app = express();
app.use(express.json());
mongoose.connect("mongodb://127.0.0.1:27017/practical5")
.then(()=>console.log("✓ Q11 MongoDB connected"))
.catch(err=>console.log("✗", err.message));
app.use("/", userRoutes);
app.listen(5011, ()=>console.log("Q11 Server running on http://localhost:5011"));

```

o/p

The screenshot shows two requests in Postman:

- POST http://localhost:5000/api/users**: A raw JSON payload is sent to create a new user. The payload is:

```

1 {
2   "name": "Hansaben Bharatbhai Parmar",
3   "email": "hbp@gmail.com",
4   "isActive": true
5 }

```

- GET http://localhost:5000/api/users/69228cae6f4a9b52d7fcfd653**: A raw JSON response is received for the created user. The response is:

```

1 {
2   "_id": "69228cae6f4a9b52d7fcfd653",
3   "name": "Hansaben Bharatbhai Parmar",
4   "email": "hbp@gmail.com",
5   "isActive": true
6 }

```

The screenshot shows two requests made using the Postman API testing tool:

**PUT Request:**

- Method: PUT
- URL: `http://localhost:5000/api/users/69228cae6f4a9b52d7fc653`
- Body tab selected.
- Body type: raw JSON.
- Request body (raw JSON):

```

1 {
2   "name": "Mehul bighra",
3   "email": "mb@gmail.com",
4   "isActive": false
5 }
6
    
```

- Response status: 200 OK
- Response time: 22 ms
- Response size: 340 B
- Response content-type: application/json
- Response body (raw JSON):

```

1 {
2   "_id": "69228cae6f4a9b52d7fc653",
3   "name": "Mehul bighra",
4   "email": "mb@gmail.com",
5   "isActive": false,
6   "__v": 0
7 }
    
```

**DELETE Request:**

- Method: DELETE
- URL: `http://localhost:5000/api/users/69228cae6f4a9b52d7fc653`
- Body tab selected.
- Body type: raw JSON.
- Request body (raw JSON):

```

1 {
2   "name": "Mehul bighra",
3   "email": "mb@gmail.com",
4   "isActive": false
5 }
6
    
```

- Response status: 200 OK
- Response time: 10 ms
- Response size: 261 B
- Response content-type: application/json
- Response body (raw JSON):

```

1 {
2   "message": "User deleted"
3 }
    
```