Implement the Student Record Management API that can be built using Node.js and MongoDB to perform CRUD operations and calculate average marks

Create Your Project Folder

mkdir student-api

cd student-api

Initialize Node.js Project

npm init -y

Install Dependencies

npm install express mongoose

Create a File (e.g., server.js)

A) CHECK CREATE STUDENT (POST)

URL:

POST http://localhost:3000/students

Body → choose **JSON**:

```
{
  "name": "Padmini",
  "rollNo": 101,
  "marks": 90,
  "course": "CS"
}
```

Click **Send**.

You will get output like:

{

```
  "_id": "648fjh74623...",
  "name": "Padmini",
  "rollNo": 101,
  "marks": 90,
  "course": "CS",
  "__v": 0
}
```


B) CHECK ALL STUDENTS (GET)

URL:

GET http://localhost:3000/students

Click **Send**.

Example output:


```
[
  {
    "_id": "648fjh74623...",
    "name": "Padmini",
    "rollNo": 101,
    "marks": 90,
    "course": "CS"
  }
]
```


C) CHECK UPDATE STUDENT (PUT)

URL:

PUT http://localhost:3000/students/<id>

(Replace <id> with the student ID returned from POST)

Body:


```
{
  "marks": 95
}
```


D) CHECK DELETE STUDENT (DELETE)
DELETE http://localhost:3000/students/<id>

Output:

{ "message": "Deleted" }

E) CHECK AVERAGE MARKS

URL:

GET http://localhost:3000/students/average

Output:

```
[
  {
    "avgMarks": 87.5
  }
]
```

Design a Node.js application to perform the following operations using the MongoDB native driver:

1. **Read** an array of objects from input.json.
2. **Connect** to MongoDB and insert the data into a database TestDB and collection items.
3. **Update** the document having id = 1 by modifying its status field.
4. **Retrieve** all documents and save them into output.json.
5. Print a success message after exporting the updated data.

Step 1: Create a Project Folder

Example folder name:

mongodb-file-ops

Open VS Code or Terminal inside this folder.

Step 2: Create Required Files

You must have **three files**:

server.js (your Node.js script)

input.json (the input data)

package.json (created automatically using npm)

Step 3: Create input.json File

Inside your folder, create a file named **input.json**:

```
[
  { "id": 1, "name": "Item A", "status": "new" },
  { "id": 2, "name": "Item B", "status": "pending" }
]
```

Step 4: Create server.js File

```
const fs = require("fs");
const { MongoClient } = require("mongodb");

async function run() {
  // 1. Read input.json
  const data = JSON.parse(fs.readFileSync("input.json"));

  // 2. Connect to MongoDB
  const client = new MongoClient("mongodb://localhost:27017");
  await client.connect();
  const db = client.db("TestDB");
  const col = db.collection("items");

  // 3. Insert JSON data into MongoDB
  await col.insertMany(data);

  // 4. Update item with id = 1
  await col.updateOne({ id: 1 }, { $set: { status: "updated" } });

  // 5. Read updated data
  const result = await col.find().toArray();

  // 6. Export updated data to output.json
```

```
  fs.writeFileSync("output.json", JSON.stringify(result, null, 2));

  console.log("✔ Exported Updated Data to output.json");

  client.close();
}

run();
```

Step 5: Install Dependencies

Run this in terminal:

```
npm init -y
npm install mongodb
```

Step 6: Start MongoDB Server

If you are using **MongoDB Community Edition**, run:

```
mongod
```

If you use **MongoDB Compass**, just keep Compass open — it auto-starts the server.

Step 7: Run the Application

Now run:

```
node server.js
```

Output

Exported Updated Data to output.json