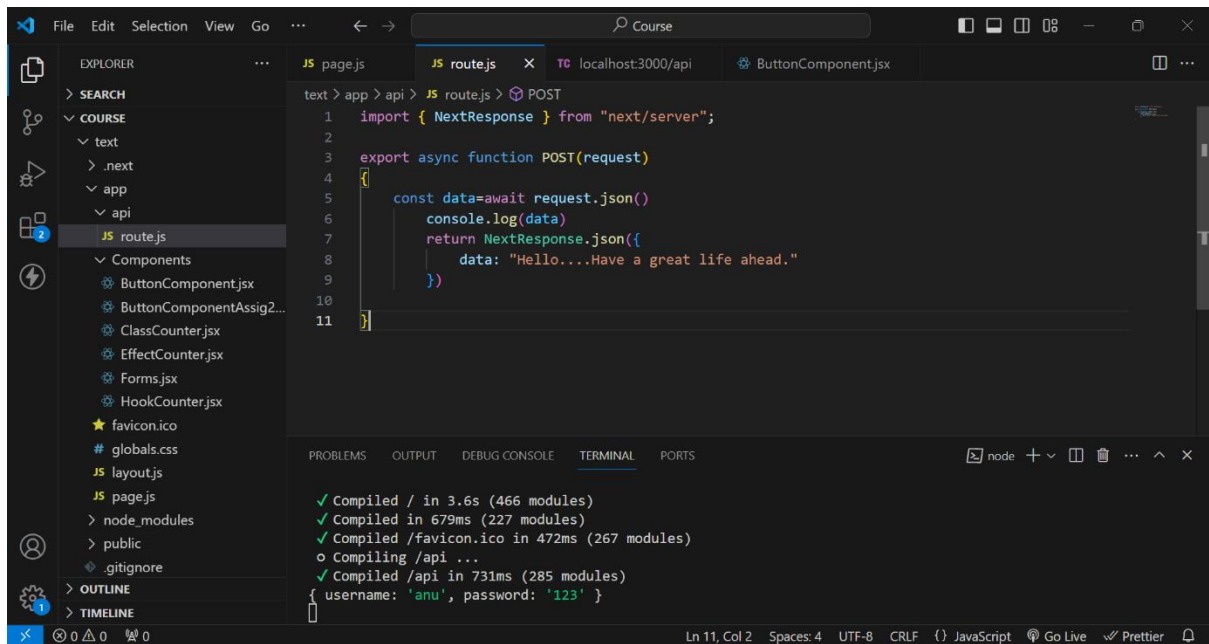


ASSIGNMENT 3- A simple RESTFUL API using React.js, DataBase Creation, and Creating Index

RestFul API code



The screenshot shows the Visual Studio Code editor with the following components:

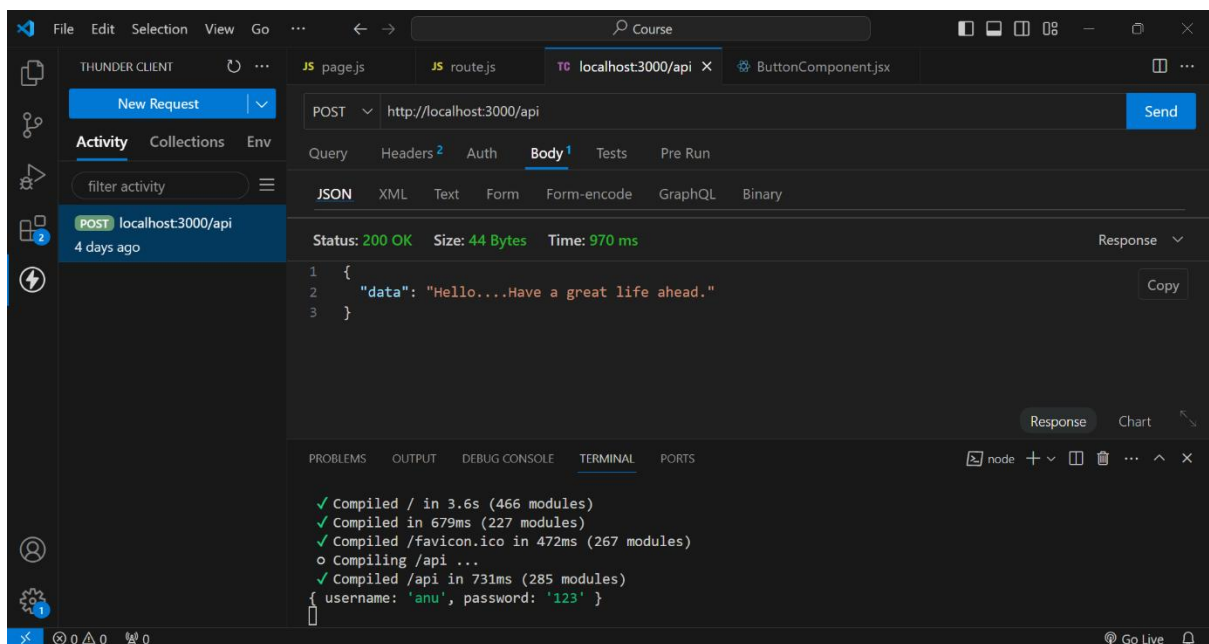
- EXPLORER:** A file tree on the left showing the project structure. The file `route.js` is selected under the `api` directory.
- EDITOR:** The main workspace displays the content of `route.js`. The code is as follows:

```
1 import { NextResponse } from "next/server";
2
3 export async function POST(request)
4 {
5   const data=await request.json()
6   console.log(data)
7   return NextResponse.json({
8     data: "Hello....Have a great life ahead."
9   })
10 }
11 }
```
- TERMINAL:** The terminal at the bottom shows the output of the development server. It indicates that the application has compiled successfully and is running on `localhost:3000`. The output includes:

```
✓ Compiled / in 3.6s (466 modules)
✓ Compiled in 679ms (227 modules)
✓ Compiled /favicon.ico in 472ms (267 modules)
o Compiling /api ...
✓ Compiled /api in 731ms (285 modules)
{ username: 'anu', password: '123' }
```

Rest Output on ThunderClient

This is the output when we create an API using REST.



The screenshot shows the Thunder Client interface with the following components:

- THUNDER CLIENT:** The top bar shows the client is running on `localhost:3000/api`.
- Activity:** A list of requests is shown. The first request is a `POST` to `localhost:3000/api`, which was executed 4 days ago.
- Request Details:** The selected request is a `POST` to `http://localhost:3000/api`. The `Body` tab is active, showing the request body in JSON format:

```
{
  "data": "Hello....Have a great life ahead."
}
```
- Response:** The response is displayed below the request body. It shows a `Status: 200 OK`, `Size: 44 Bytes`, and `Time: 970 ms`. The response body is in JSON format:

```
{
  "data": "Hello....Have a great life ahead."
}
```
- TERMINAL:** The terminal at the bottom shows the output of the development server, which is identical to the one in the first screenshot.

DataBase Creation

1st we created a database named test

In that test we created a collection named Students

And finally by using ADD DATA , we added data of the students like rno, name, age.

The screenshot shows the MongoDB Compass interface for the 'test' database, specifically the 'Students' collection. The 'Documents' tab is selected, showing a table with 7 documents. The table has columns: _id, ObjectId, RNo, Int32, Name, String, and Age, Int32. The data is as follows:

_id	ObjectId	RNo	Int32	Name	String	Age	Int32
1	ObjectId('65ebebfc359cf2...')	1		"ABC"		20	
2	ObjectId('65ebec56d359cf2...')	2		"DEF"		19	
3	ObjectId('65ebecd6d359cf2...')	3		"AHF"		20	
4	ObjectId('65ebed18d359cf2...')	4		"DER"		22	
5	ObjectId('65ebed48d359cf2...')	5		"UDE"		20	
6	ObjectId('65ebed67d359cf2...')	6		"SPR"		18	

Indexing Creation

As rno is the unique attribute through which we identify a student uniquely. So, here I created an index based on Rno. And assigned unique key to it.

The screenshot shows the MongoDB Compass interface for the 'test' database, specifically the 'Students' collection. The 'Indexes' tab is selected, showing a table with 2 indexes. The table has columns: Name and Definition, Type, Size, Usage, and Properties. The data is as follows:

Name and Definition	Type	Size	Usage	Properties
> RNo_1	REGULAR	36.9 KB	0 (since Sat Mar 09 2024)	UNIQUE
> _id_	REGULAR	36.9 KB	29 (since Sat Mar 09 2024)	UNIQUE

Indexing usage

As we gave Rno as unique, If we try to insert a duplicate document that is existed it shows an error like..duplicate collection.

The screenshot shows the MongoDB Compass interface for the 'test' database, specifically the 'Students' collection. The interface displays 7 documents and 2 indexes. The 'Documents' tab is active, showing a table of student records. The table has columns: _id (ObjectId), RNo (Int32), Name (String), and Age (Int32). The data is as follows:

	_id ObjectId	RNo Int32	Name String	Age Int32
3	ObjectId('65ebeca6a359cf2...')	3	"AHF"	20
4	ObjectId('65ebed18d359cf2...')	4	"DER"	22
5	ObjectId('65ebed48d359cf2...')	5	"UDE"	20
6	ObjectId('65ebed67d359cf2...')	6	"SPR"	18
7	ObjectId('65ebec2ad359cf2...')	6	No field	No field

A red error message is displayed at the bottom of the table:

Plan executor error during findAndModify :: caused by :: E11000 duplicate key error coll...

The error message is accompanied by 'CANCEL' and 'UPDATE' buttons. The bottom status bar shows '>_MONGOSH'.