Next JS - 3

Create User Data

In this topic, we will try to use Mongo DB to store our own User information.

The user data consists of:

- id: Mongo DB ID
- username: string
- email: string
- password: string (hashed)
- firstname: string
- lastname: string
- status: [“ACTIVE”, “SUSPENDED”, “DELETED”]

Unique key in Mongo

Before we create our first POST method to insert new users, let’s add unique keys into our collection.

If you want to set a field as unique, you need to set it as an index and set its “unique” property to “true”. This process should be done explicitly, or just at system initialization step.

You may create an Administrator API call which may run by Administrator during initiation.

The following example creates an API with some protection. This API will initial collection index in our Mongo DB.

Simple Admin Initial Index:

1. Create a file named “ensureIndexes.js” in “src/lib”:

```
import { getClientPromise } from "@/lib/mongodb";

export async function ensureIndexes() {
  const client = await getClientPromise();

  const db = client.db("wad-01");
  const userCollection = db.collection("user");
  await userCollection.createIndex({ username: 1 }, { unique: true });
  await userCollection.createIndex({ email: 1 }, { unique: true });
}
```

This code will set the field username and email as indexes with unique value.
This will ensure protecting duplicate username and email.
You can also apply this code for every collection.
The unique shall be set before you insert any data to prevent duplication.

2. Create a new directory “admin” under “api”
3. Create a new directory “initial” under “api/admin”
4. Create a file named route.js with the code:

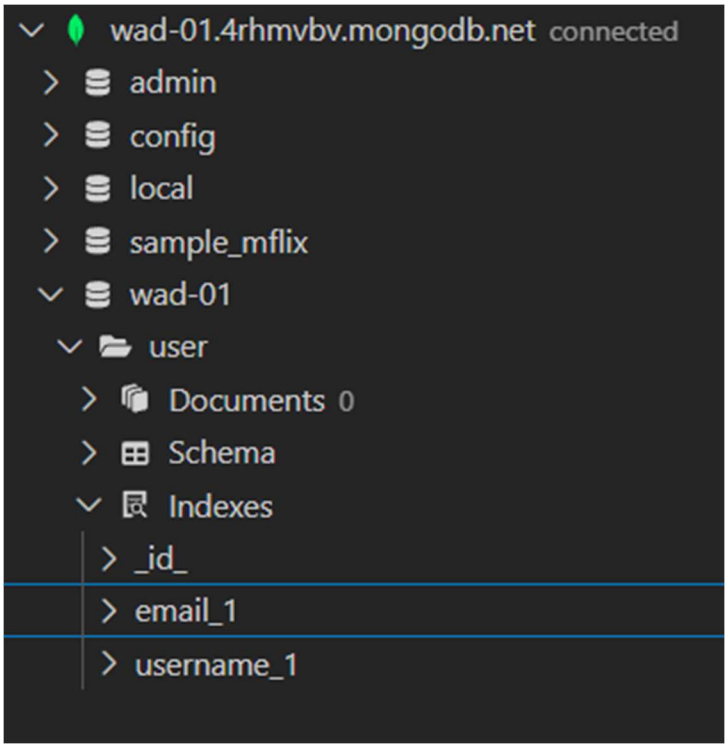
```
import { ensureIndexes } from "@/lib/ensureIndexes";
import { NextResponse } from "next/server";

export async function GET(request) {

  const { searchParams } = new URL(request.url);
  const challenge = searchParams.get("pass") ?? false;

  if (!challenge) {
    return NextResponse.json({
      message: "Invalid usage"
    }, {
      status: 400
    })
  }

  const pass = process.env.ADMIN_SETUP_PASS;

  if (challenge != pass) {
    return NextResponse.json({
      message: "Admin password incorrect"
    }, {
      status: 400
    })
  }

  const result = await ensureIndexes();
  return NextResponse.json({ message: "Indexes ensured" });
}
```

5. Add the following line into “.env.local” and replace the password with your credential.

```
ADMIN_SETUP_PASS=password
```

6. Start the Next server
7. Using any API tool or just browser to http://localhost:3000/admin/initial?pass=password
   Replace password with the correct password.

If you have done everything correctly, you shall see something like this



Running the initial again does not harm your Mongo DB database.

The system will throw errors if you try to create unique index to the field with already have duplicated value.

Insert User with POST method API

The following example demonstrates how to create a POST method API.

The API path is “api/user”.

1. Install bcrypt dependency, if not installed.

```
npm install bcrypt
```

2. Create a new directory “user” under “api”
3. Create a new file route.js under “user” with the following code:

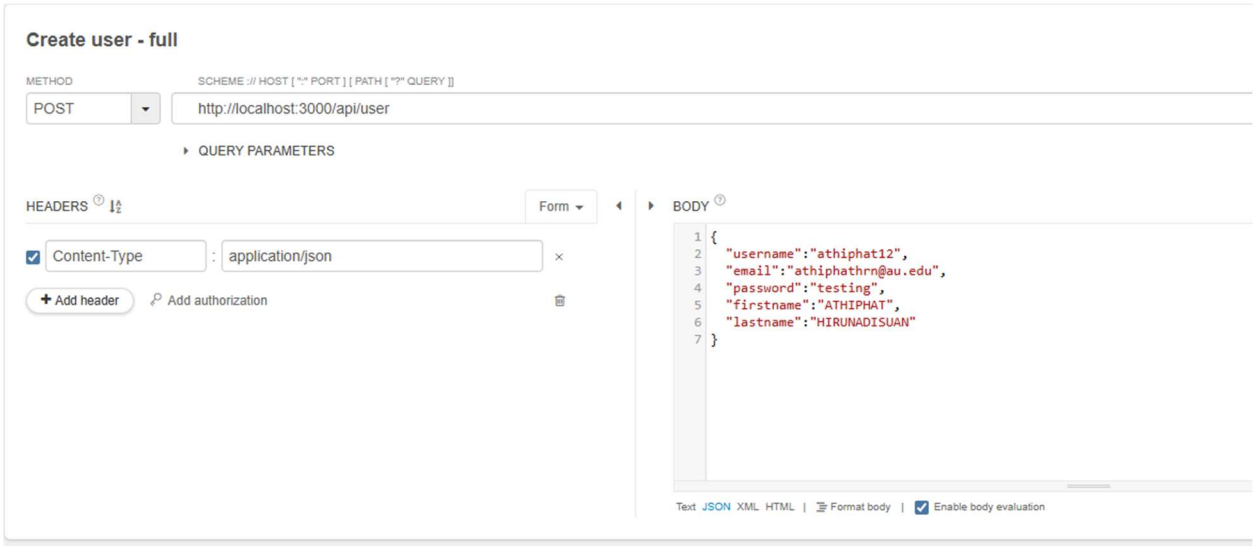
```
import corsHeaders from "@/lib/cors";
import { getClientPromise } from "@/lib/mongodb";
import bcrypt from "bcrypt";
import { NextResponse } from "next/server";

export async function  POST (req) {
  const data = await req.json();
  const username = data.username;
  const email = data.email;
```

```
const password = data.password;
const firstname = data.firstname;
const lastname = data.lastname;

if (!username || !email || !password) {
  return NextResponse.json({
    message: "Missing mandatory data"
  }, {
    status: 400,
    headers: corsHeaders
  })
}

try {
  const client = await getClientPromise();

  const db = client.db("wad-01");
  const result = await db.collection("user").insertOne({
    username: username,
    email: email,
    password: await bcrypt.hash(password, 10),
    firstname: firstname,
    lastname: lastname,
    status: "ACTIVE"
  });
  console.log("result", result);
  return NextResponse.json({
    id: result.insertedId
  }, {
    status: 200,
    headers: corsHeaders
  });
}
catch (exception) {
  console.log("exception", exception.toString());
  const errorMsg = exception.toString();
  let displayErrorMsg = "";
  if (errorMsg.includes("duplicate")) {
    if (errorMsg.includes("username")) {
      displayErrorMsg = "Duplicate Username!!"
    }
    else if (errorMsg.includes("email")) {
      displayErrorMsg = "Duplicate Email!!"
    }
  }
```

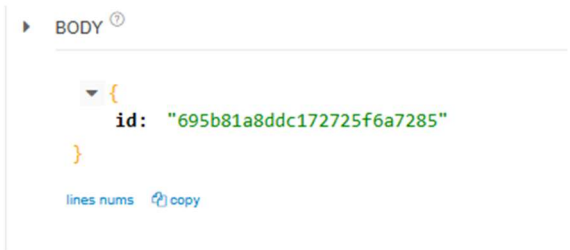
```
    return NextResponse.json({
      message: displayErrorMsg
    }, {
      status: 400,
      headers: corsHeaders
    })
  }

}
```

4. Start the Next JS (if stopped).
5. Use any API tool to submit with POST method and proper user data in JSON format, for example:

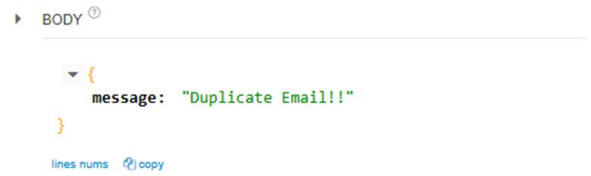
```
{
 "username":"athiphat12",
 "email":"athiphathrn12@au.edu",
 "password":"testing",
 "firstname":"ATHIPHAT",
 "lastname":"HIRUNADISUAN"
}
```



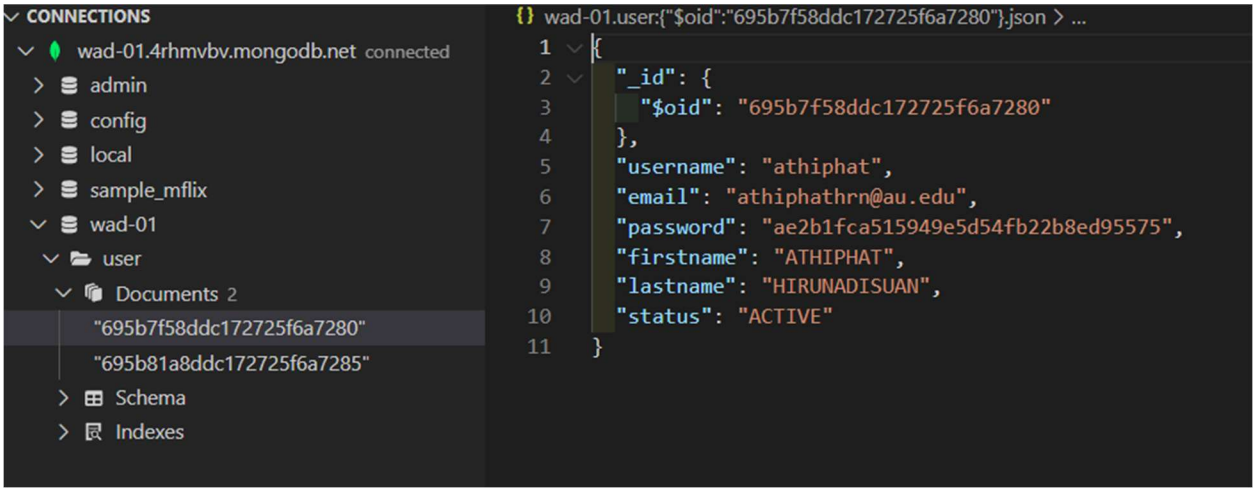
6. For success return:



7. For duplication error return:



8. Observe data via VS Code



Assignment 4 – User Management page