How to Connect a Next.js App to a MongoDB Database

Introduction:

This guide will walk you through the steps needed to connect a Next.js app to a MongoDB database using Mongoose, a popular Object Data Modeling (ODM) library for MongoDB and Node.js. By the end of this guide, you will be able to create, read, update, and delete (CRUD) data in MongoDB directly from your Next.js application. This guide assumes that you already have a basic Next.js project set up.

Tools Needed:

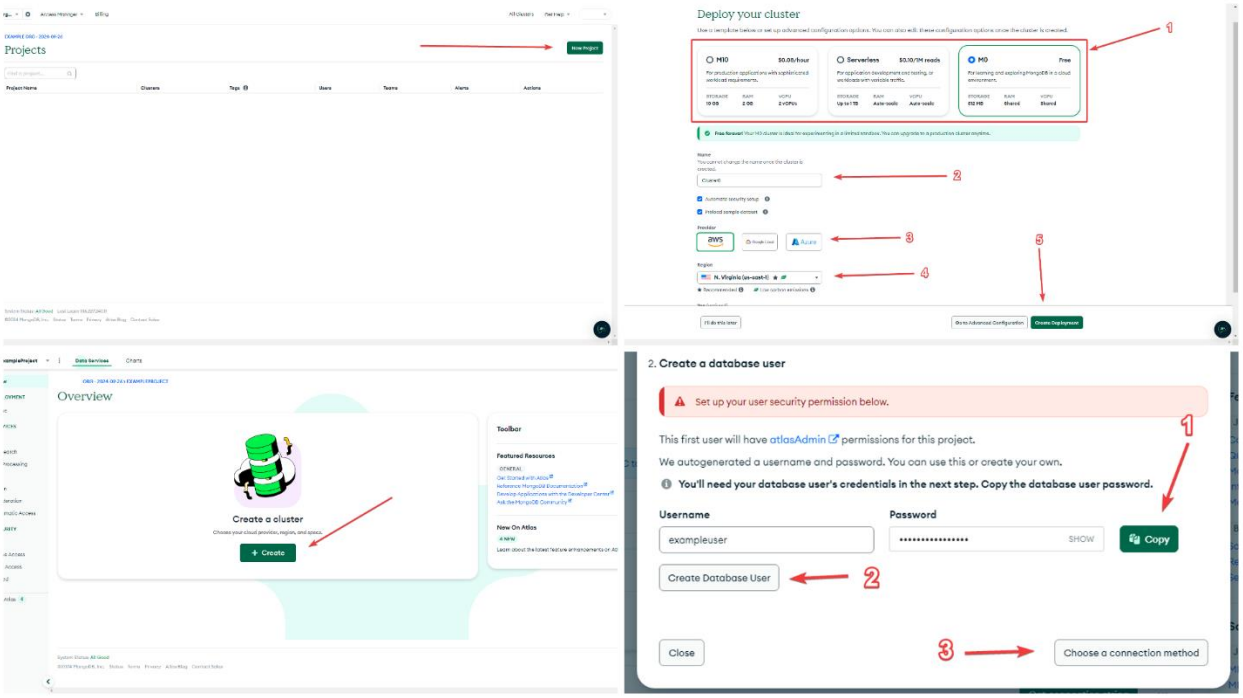
- A Next.js project (version 12 or later recommended)
- MongoDB Atlas account (or a local MongoDB instance)
- Node.js and npm installed
- A text editor (VS Code recommended)
- Internet connection

Steps:

1. **Set up MongoDB Atlas or Local MongoDB:**
    - If using MongoDB Atlas (recommended for beginners):
        1. Sign in to your MongoDB Atlas account at mongodb.com.
        2. If a project has not been created yet. Please create one by clicking the “New Project” button and inserting a project name.
        3. Create a new cluster by clicking the green “Create” button.
        4. Select the options that fits your needs, insert a name for your cluster and click the green button that says, “Create Deployment.”

5. Once the cluster is ready, make sure to store the database user information, click “Create Database User”, then click “Choose a connection method.”
6. Select “Drivers” and choose “Mongoose” as the driver and save the connection string provided in Step 3 of the pop up.

- If using a local MongoDB instance:
    1. Download and install MongoDB locally from the official website.
    2. Start your local MongoDB server by running the mongod command in your terminal.



2. Install Mongoose in your Next.js Project:

1. Open your terminal and navigate to your Next.js project folder.
2. Run the following command to install Mongoose.

```
D:\MyNextJSProject>npm install mongoose
```

3. Create a Database Connection File

1. Inside your Next.js project, create a *lib* folder in the root directory.
2. Create a new file called *database.js* inside the *lib* folder.
3. Add the following template code to your file:

```
import mongoose from 'mongoose';


const MONGODB_URI = process.env.MONGODB_URI;


if (!MONGODB_URI) {
    throw new Error('Please define the MONGODB_URI environment variable inside .env.local');
}
let cached = global.mongoose;
if (!cached) {
    cached = global.mongoose = { conn: null, promise: null };
}
async function connectToDatabase() {
    if (cached.conn) {
        return cached.conn;
    }
    if (!cached.promise) {
        const opts = {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        };
        cached.promise = mongoose.connect(MONGODB_URI, opts).then((mongoose) => {
            return mongoose;
        });
    }
    cached.conn = await cached.promise;
    return cached.conn;
}
export default connectToDatabase;
```

4. Set up Environment Variables:

1. Create a *.env.local* file in the root of your Next.js project.

2. Add the following line, replacing the text with your MongoDB connection string, and adding the password:

```
MONGODB_URI=”Your MongoDB Connection String Here”
```

5. Create a Mongoose Schema and Model:

1. Inside the *lib* folder, create a new file called models.js.

2. Define a schema for the data you want to store. For example, a simple user schema:

```
import mongoose from 'mongoose';


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


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


export default User;
```

6. Create an API Route to Connect to the Database:

1. Inside the *pages/api* folder, create a new file called *users.js*.
2. Add the following code to connect to the database and create a new user:

```
import connectToDatabase from '../../lib/database';
import User from '../../lib/models';

export default async function handler(req, res) {
  await connectToDatabase();

  if (req.method === 'POST') {
    const { name, email } = req.body;
    try {
      const newUser = new User({ name, email });
      await newUser.save();
      res.status(201).json({ success: true, data: newUser });
    } catch (error) {
      res.status(400).json({ success: false, message: error.message });
    }
  }
}
```

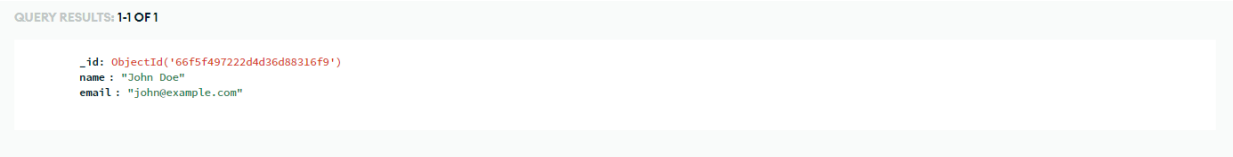
Note: This code makes is so when a post request with a body containing a name and an email is made to the URL /api/users, it adds the user to the database.

7. Test Your API and Database Connection:

1. Run your Next.js application by executing: npm run dev.
2. Use a tool like Postman or curl to send a *POST* request to http://localhost:3000/api/users with a JSON body:

```
{
   "name": "John Doe",
   "email": "john@example.com"
}
```

3. If successful, a new user will be added to your MongoDB database.



Conclusion:

By following these steps, you have successfully connected your Next.js application to a MongoDB database using Mongoose. This setup allows you to perform CRUD operations and scale your database as needed. You can now proceed to build more complex models, set up authentication or optimize your API routes.

Glossary:

- **MongoDB Atlas:** A cloud-based MongoDB service for easy database hosting.
- **ODM:** Object Data Modeling, a tool to manage data using JavaScript objects and MongoDB.
- **Schema:** A blueprint for defining the structure of documents in MongoDB
- **API Route:** A function that handles HTTP requests in Next.js.

Warnings:

- Never share your MongoDB connection string publicly. Keep it secure in environment variables.
- Ensure MongoDB is configured properly to avoid downtime or errors.

Appendix

- **Useful Resources**
  - MongoDB Official Docs
  - Mongoose Documentation