## Setting up connection in mongodb with nodejs

Goto <https://mongodb.com/cloud/atlas>

Then create a database with node drivers selected

Copy the code with username and password

//importings

const express = require("express");

const app = express();

const PORT = 5000;const mongoose = require("mongoose");

//this contains all MONGOURL object that is our URL copied form

const {MONGOURI} = require("./valuekeys.js");

// connection part --=========================================================

// it is just like mysqli\_connect in php with url that we used to do with jdbc

mongoose.connect(MONGOURI);

// when connection function returns connected then log the console with the message

mongoose.connection.on("connected",()=>{

    console.log("we are connected to the mongodb database")

});

// same as die("error") in php

mongoose.connection.on("error",()=>{

    console.log("unsuccessful connecting to the mongodb database")

});

app.get("/", (req, res) => {

    res.send("Hello World");

});

// Start the server

app.listen(PORT, () => {

    console.log("Server is running on [http://localhost](http://localhost/):" + PORT);

});

# Creating a user schema

This is just like how we define table in MYSQL with constraints

Remember that SQL had definite query structure but in mongodb everytihing is given as JSON creating entity object

const mongoose = require("mongoose");

// creating a table like structure for database like we do in mysql

const userSchema  = new mongoose.Schema({

    name:{

        type:String,

        required: true

    },

    email:{

        type:String,

        required: true,

        unique: true

    },

    password:{

        type:String,

        required: true

    }

})

//importing that schema with the name User

module.exports = mongoose.model("User", userSchema);

## **Using route endpoints to signin and signup the user (you need to use postman or thunderclient for this)**

const express = require("express");

const router = express.Router();

const mongoose = require("mongoose");

const User = mongoose.model("User");

const bcrypt = require("bcryptjs");

// Define your secret pepper value

const pepper = "bazinga";

// Define your secret pepper value

router.post("/signup", (req, res) => {

const { name, email, password } = req.body;

if (!name || !email || !password) {

return res.status(422).json({ error: "Please provide all the information" });

}

// Check if the email already exists in the database

User.findOne({ email: email })

.then(existingUser => {

if (existingUser) {

return res.status(422).json({ error: "Email already exists" });

}

// Hash the raw password

bcrypt.hash(password, 10)

.then(hashedPassword => {

// Create a new user with the hashed password

const newUser = new User({

email, password: hashedPassword, name

});

// Save the user to the database

newUser.save()

.then(user => {

res.json({ message: "User registered successfully" });

})

.catch(err => {

console.error("Error saving user:", err);

res.status(500).json({ error: "Failed to register user" });

});

})

.catch(err => {

console.error("Error hashing password:", err);

res.status(500).json({ error: "Failed to register user" });

});

})

.catch(err => {

console.error("Error finding existing user:", err);

res.status(500).json({ error: "Failed to register user" });

});

});

router.get("/", (req, res) => {

res.send("User authentication site");

});

module.exports = router;

router.post("/signin", (req, res) => {

// Taking user-given email and password by destructuring

const { email, password } = req.body;

if (!email || !password) {

return res.status(422).json({ error: "Please enter email and password" });

}

// Find the user by email in the database

User.findOne({ email: email })

.then(savedUser => {

if (!savedUser) {

return res.status(422).json({ error: "Invalid email" });

}

// Hash the raw password with the pepper and compare it with the hashed password stored in the database

bcrypt.compare(password, savedUser.password)

.then(matched => {

if (!matched) {

return res.status(422).json({ error: "Invalid password" });

}

res.status(200).json({ message: "Signed in" });

})

.catch(err => {

console.error("Error comparing passwords:", err);

res.status(500).json({ error: "Failed to sign in" });

});

})

.catch(err => {

console.error("Error finding user:", err);

res.status(500).json({ error: "Failed to sign in" });

});

});

router.get("/", (req, res) => {

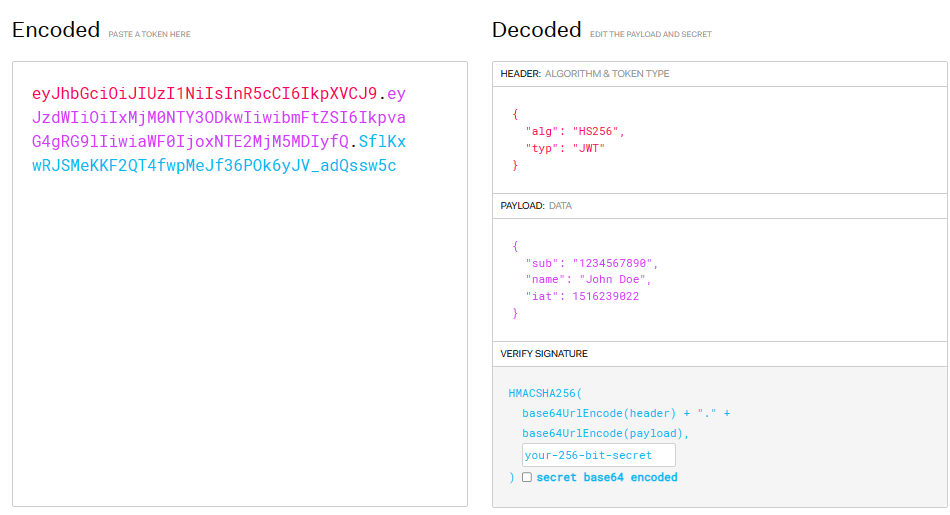
res.send("User authentication site");

});

module.exports = router;

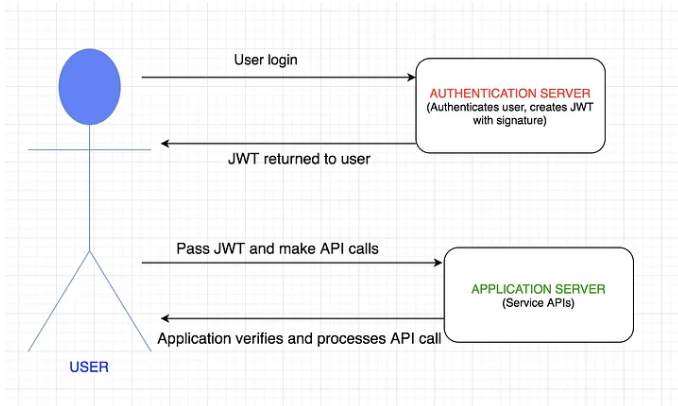
# JSON webtoken

JSON Web Token (JWT) is a JSON based method of transferring data between two parties in a compact, self-contained and secure manner. It is an open standard (RFC 7519) used especially in the context of web browser single sign-on (SSO).



## Working of JSON

SON Web Tokens (JWTs) are used for authentication in client-server communication. After a user logs in, the authentication server creates a signed JWT containing user information. The client then includes this JWT in API calls to access resources. The application server verifies the JWT's authenticity using a secret key or a public/private key pair. If the signatures match, trust is established between the client and the server, allowing secure communication.



## Where to use JWT?

1. Access to your JSON Web Token (JWT) could potentially allow unauthorized users to access your data.
2. JWTs are encoded but not encrypted, so their contents can be decoded.
   1. To mitigate this risk:

* Use short-lived JWTs.
* Utilize HTTPS encryption for communication.
* Implement mechanisms for token revocation.
* Minimize sensitive data stored within JWTs.
* Consider additional security layers like IP restriction and multi-factor authentication.

1. While the risk exists, proper security measures can significantly reduce the impact of token theft.

JWT in action

>>npm i jsonwebtoken