JSON WEB TOKENS

1. **User Login:**
   - The user sends their credentials (username and password) to the server.
2. **Server Verification:**
   - The server verifies the credentials.
   - If the credentials are correct, the server generates a JWT and sends it back to the client
3. **Client Storage:**
   - The client stores the JWT (usually in local storage or a cookie).
4. **Subsequent Requests:**
   - For subsequent requests, the client includes the JWT in the Authorization header.
   - The server verifies the JWT and processes the request if the JWT is valid.

Creating server→main.js

```
const express = require("express");
const app = express();
const PORT = 4000;

app.listen(PORT, () => {
  console.log(`Server started and running @ ${PORT}`);
});
```

```
[nodemon] starting `node main.js`
Server started and running @ 4000
```

To work with jsonwebtokens we have to install→

```
PS G:\MERN STACK\jwt tokens> npm install jsonwebtoken
```

And require it -→

```
const express = require("express");
const jwt = require("jsonwebtoken");
const app = express();
const PORT = 4000;

app.listen(PORT, () => {
  console.log(`Server started and running @ ${PORT}`);
});
```

STEP –1 CREATING USERS (API)

```
const express = require("express");
const jwt = require("jsonwebtoken");
const app = express();
const users = [
  {
    id: "1",
    username: "mahesh",
    password: "mahesh",
    isAdmin: true, // admin have all permissions
  },
  {
    id: "2",
    username: "suresh",
    password: "suresh",
    isAdmin: false,
  },
];
const PORT = 4000;

app.listen(PORT, () => {
  console.log(`Server started and running @ ${PORT}`);
});
```

```
//express.json() middleware is used for parsing JSON request bodies.
app.use(express.json())
```

Step – 2 Now creating route for login for users

```
app.post("/api/login", (req, res) => {
  const { username, password } = req.body; // data coming from client


  const user = users.find((person) => {
    return person.username === username && person.password === password;
  });
```

explaining each step below→

1. **Extracting Data from the Request Body:**

```javascript
const { username, password } = req.body;
```

- This line extracts `username` and `password` from `req.body`.
- `req.body` contains the data sent by the client in the body of the POST request.
- The client sends a request with JSON data like:

```json
{
  "username": "mahesh",
  "password": "mahesh"
}
```

- Using destructuring assignment, `const { username, password } = req.body;` assigns the values of `username` and `password` from the request body to the variables `username` and `password`.

2. **Finding the User in the Array:**

```javascript
const user = users.find((person) => {
  return person.username === username && person.password === password;
});
```

- `users` is an array of user objects:

```javascript
const users = [
  {
    id: "1",
    username: "mahesh",
    password: "mahesh",
    isAdmin: true,
  },
  {
    id: "2",
    username: "suresh",
    password: "suresh",
    isAdmin: false,
  },
];
```

- `users.find(...)` is a method that iterates through the `users` array and returns the first element that satisfies the provided testing function.

- The testing function is defined as:

```javascript
(person) => {
  return person.username === username && person.password === password;
}
```

  - `person` is each user object in the array during the iteration.
  - `person.username === username`: This checks if the `username` of the current `person` matches the `username` extracted from `req.body`.
  - `person.password === password`: This checks if the `password` of the current `person` matches the `password` extracted from `req.body`.
- If a user with matching `username` and `password` is found, the `find` method returns that user object.
- If no matching user is found, the `find` method returns `undefined`.

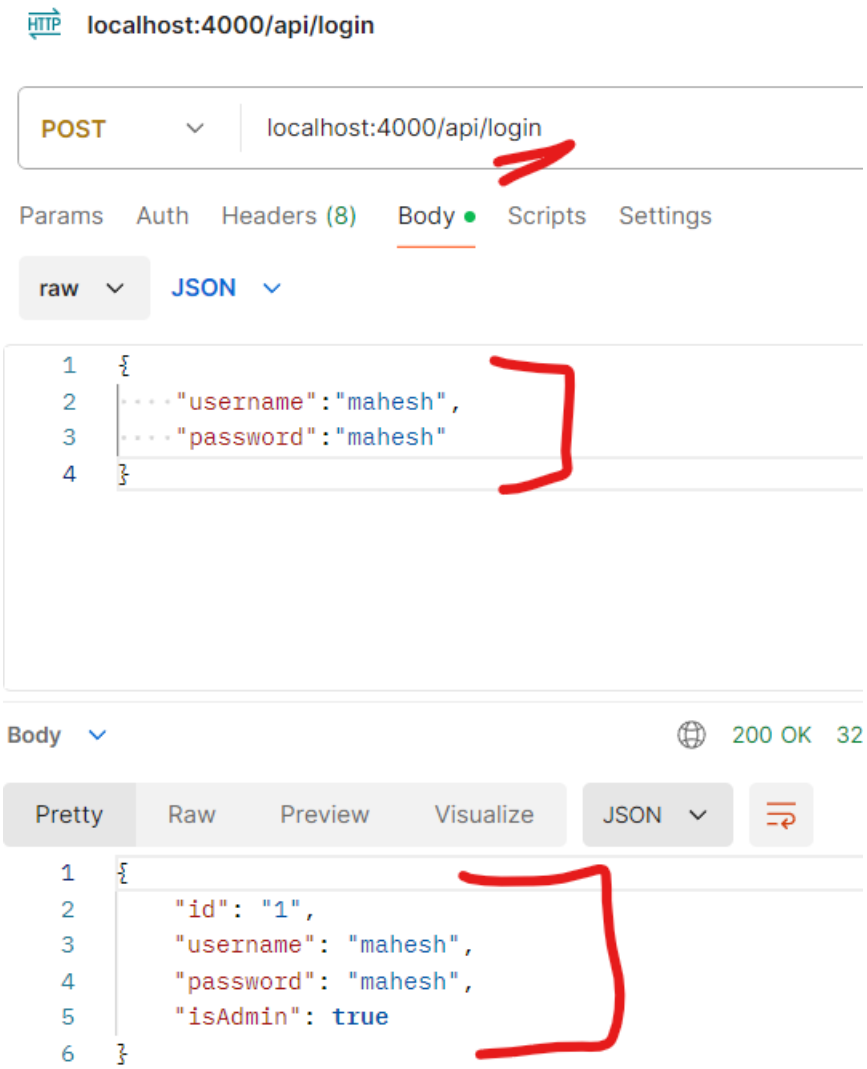
If person found that user details will assign to user varible

If user found we have to convert in json format other wise we have return error message

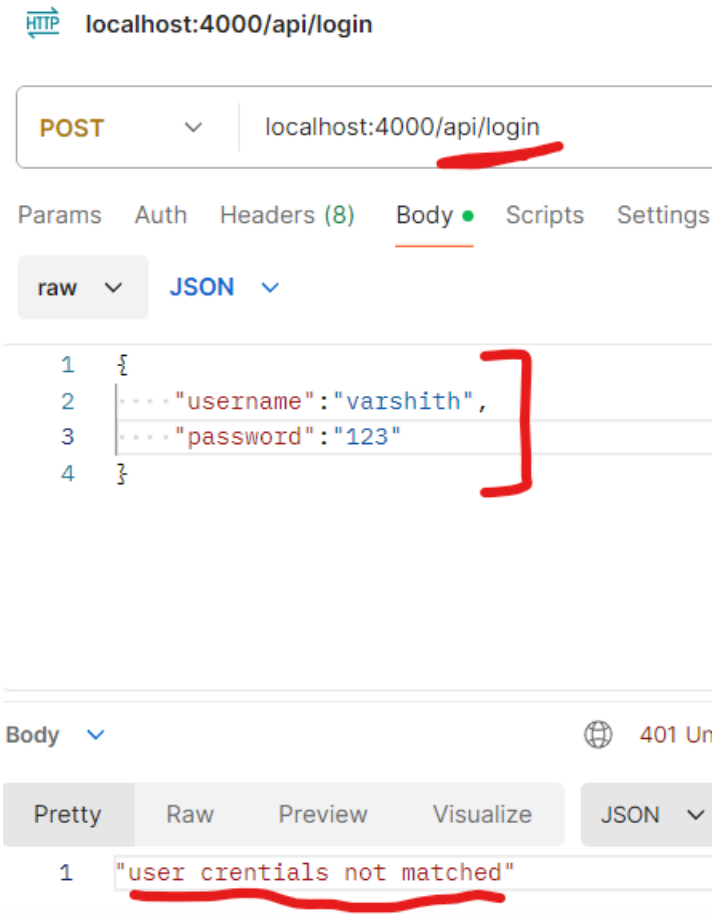
```javascript
app.post("/api/login", (req, res) => {
  const { username, password } = req.body; // data coming from client


  const user = users.find((person) => {
    return person.username === username && person.password === password;
  });
  if(user)
  {
      res.json(user)
    }
    else{
      res.status(401).json("user crentials not matched")
    }
});
```

STEP-4 →Now we will check by login (WITH DIFFERENT CRENDITALS) in (using postman your api is working or not)

If we are login with correct username and password



If we are trying to login with incorrect usename and password which is not present in users array



Step-5 → jwt authecation implementation

Convert user details into token after login with correct username and password

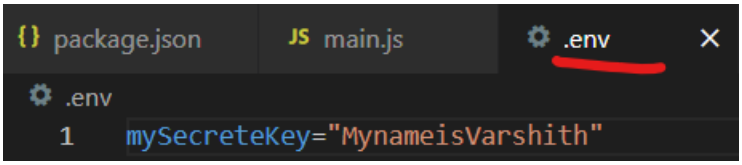
➔ jwt.sign() used to convert userdetails to token

** jwt.sign() → takes 2 arguments

1) argument 1 → by using which details token should be generated

2) argument 2 → secret key → we can write anything but keep secretly and remember it

Ex→ in env




```
8  const secretKey = process.env.mySecreteKey;
```

Creation of token according to details provided

```
app.post("/api/login", (req, res) => {
  const { username, password } = req.body; // data coming from client

  const user = users.find((person) => {
    return person.username === username && person.password === password;
  });
  if (user) {
    const accessToken = jwt.sign(
      {
        id: user.id,
        isAdmin: user.isAdmin,
      },
      secretKey
    );
    res.json({
      username: user.username,
      isAdmin: user.isAdmin,
      accessToken,
    });
  } else {
    res.status(401).json("user crentials not matched");
  }
});
```

To see token go to postman check by providing correct username and password

localhost:4000/api/login

Save

POST localhost:4000/api/login

Params Auth Headers (8) Body Scripts Settings

raw JSON

```
1 {
2     "username":"suresh",
3     "password":"suresh"
4 }
```

Body 200 OK 404 ms 429 B Save

Pretty Raw Preview Visualize JSON

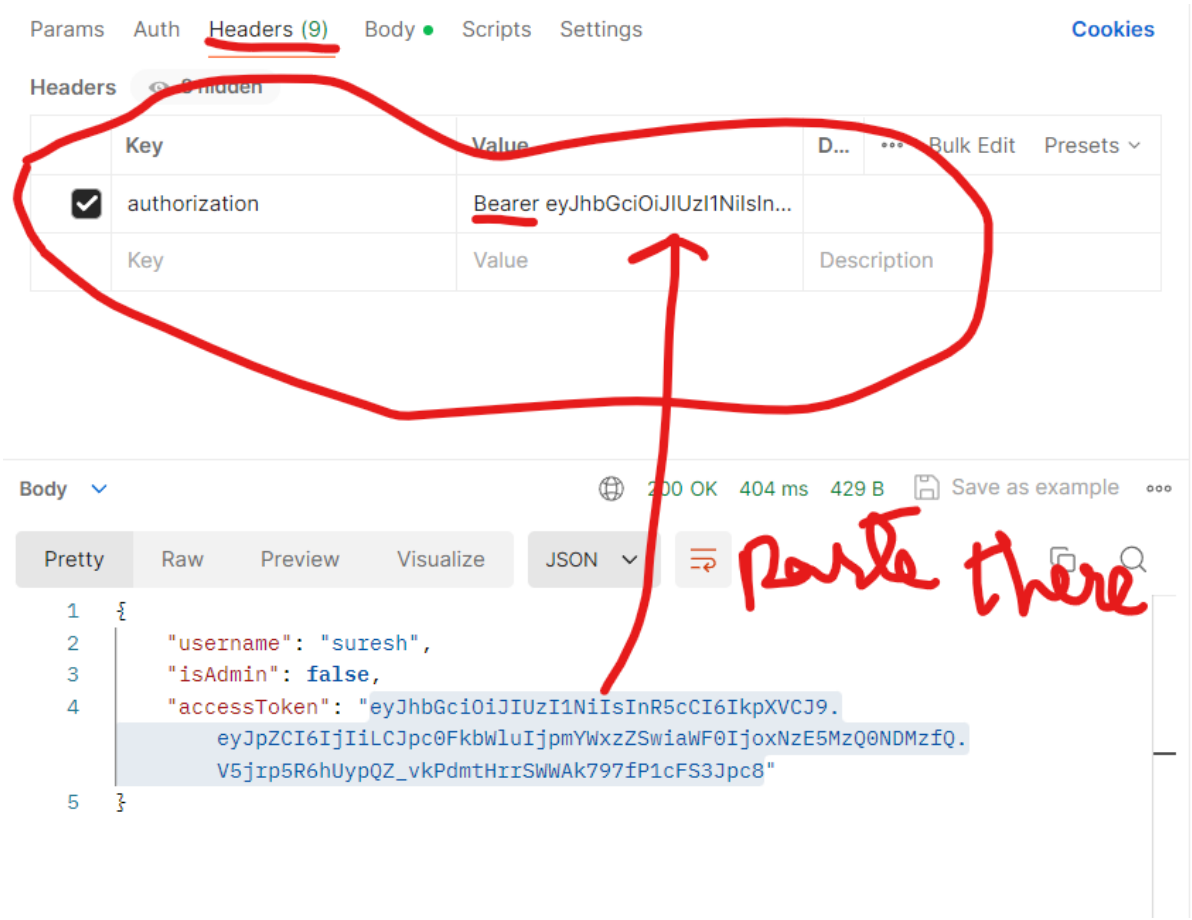

```
1 {
2     "username": "suresh",
3     "isAdmin": false,
4     "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6IjIiLCJpc0FkbWluIjpmYWxzZSwiaWF0IjoxNzE5MzQ0NDMzfQ.
        V5jrp5R6hUypQZ_vkPdmtHrrSWWAk797fP1cFS3Jpc8"
5 }
```

Step – 6
now the task is →

**By using their individual accesstokens → they can delete their account
Mahesh is admin so he can delete any account because he is admin
but , suresh can only delete his account because he is not an admin

-→now we will create a delete route and learn by using jwt tokens how can we perform operations

→jwt will present in header so,



A **bearer token** is a type of access token that is typically issued by an authorization server. The token is considered a "bearer" token because the possession of the token itself is sufficient to gain access to a protected resource. In other words, whoever holds the token can access the resource without any additional proof of identity.

```
//creating middleware
const verifyUser = (req, res, next) => {
  // token is present in the header
  // to get it
  const userToken = req.headers.authorization;
  if (userToken) {
    // removing space and bearer and only taking token
    const token = userToken.split(" ")[1];
    //verify usertoken and pass secretkey
    jwt.verify(token, secretkey, (err, user) => {
      if (err) {
        return res.status(403).json({ err: "token is not valid" });
      }
      req.user = user;
      next();
    });
  } else {
    res.status(401).json("you are not authenticated");
  }
};
```

```
app.delete("/api/users/:userId", verifyUser, (req, res) => {
  //checking by id or isadmin
  //params.userId --> value taking from above route
  if (req.user.id === req.params.userId || req.user.isAdmin) {
    res.status(200).json("user is deleted successfull");
  }
  else{
    res.status(401).json("you are are not allowed to delete")
  }
});
```

localhost:4000/api/users/1

Save | Share

DELETE | localhost:4000/api/users/1 | Send

Params | Auth | Headers (7) | Body | Scripts | Settings | Cookies

Headers | 6 hidden

	Key	Value	D...
☑	authorization	Bearer eyJhbGciOiJIUzI1NiIsIn...	
	Key	Value	Description

Bulk Edit | Presets

Body | 200 OK | 19 ms | 264 B | Save as example

Pretty | Raw | Preview | Visualize | JSON

```
1  "user is deleted successfull"
```

Here I have passed accesskey of mahesh

DELETE localhost:4000/api/users/1 Send

Params Auth Headers (7) Body Scripts Settings Cookies

Headers 6 hidden

	Key	Value	D...
☑	authorization	Bearer eyJhbGciOiJIUzI1NiIsIn...	
	Key	Value	Description

Bulk Edit Presets

Body 401 Unauthorized 7 ms 280 B Save as example

Pretty Raw Preview Visualize JSON

```
1 "you are are not allowed to delete"
```

here not allowed to delete user 1 beecause here I have passed accessToken of suresh where suresh is not admin

➔ but we can delete user 2

DELETE localhost:4000/api/users/2 Send

Params Auth Headers (7) Body Scripts Settings Cookie

Headers 6 hidden

	Key	Value	D...
☑	authorization	Bearer eyJhbGciOiJIUzI1NiIsIn...	
	Key	Value	Description

Bulk Edit Presets

Body 200 OK 7 ms 264 B Save as example

Pretty Raw Preview Visualize JSON

```
1 "user is deleted successfull"
```

here accessToke is suresh and user 2 also suresh we can delete

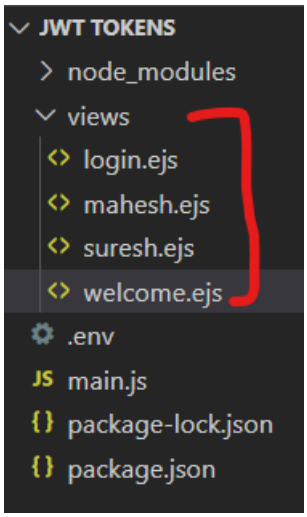
Step 7 →

Create fortend → ejs

```
PS G:\MERN STACK\jwt tokens> npm install ejs
```

```
// we are geting details from form so this middle ware is used
app.use(express.urlencoded({ extended: true }));
```

Create views folder in it create ejs files-→



Login.ejs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>User Login</h1>
    <form action="/api/login" method="POST">
      <input type="text" name="username" placeholder="Username" />
      <input type="text" name="username" placeholder="Password" />
      <button type="submit">Login</button>
    </form>
  </body>
</html>
```

Mahesh.ejs

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>Hi , I am Mahesh</h1>
    <form action="/api/logout">
      <button type="submit">Logout</button>
    </form>
  </body>
</html>
```

(Ctrl)

Suresh.ejs

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>Hi , I am Suresh</h1>
    <form action="/api/logout">
      <button type="submit">Logout</button>
    </form>
  </body>
</html>
```

(Ctrl)

Welcome.ejs

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>Welcome to My Application</h1>
  </body>
</html>
```

→ route for Mahesh /mahesh

```bash
//creating route for mahesh
app.get("/mahesh", (req, res) => {
  res.render("mahesh");
});
```

→ route for Suresh /suresh

```bash
//creating route for suresh
app.get("/suresh", (req, res) => {
  res.render("suresh");
});
```

-→ creating route for login in server.js

```
//creating route for login for login.ejs
//route should be same as in login.ejs
app.get("/api/login/:userId", (req, res) => {
  const userId = req.params.userId;
  if (userId)
  {
    if (userId === "1") {
      res.redirect("/mahesh");
    }
    else if(userId=="2")
      {
        res.redirect("/suresh");
      }
  }

  else {
    res.status(403).json("user not found")
  }
});
```

route of mahesh

route of suresh

→ creating route for logout in server.js → no actual it just to cut token

```
//craeting route for logout
//for log out we have to cut the token link so we written post
app.post("/api/logout", (req, res) => {
  const userToken = req.headers.authorization;
  if (userToken) {
    const token = userToken.split(" ")[1];
    if (token) {
      //storing tokens in array
      let allTokens = [];
      const tokenIndex = allTokens.indexOf(token);
      if (tokenIndex !== -1) {
        allTokens.splice(tokenIndex, 1);
        res.status(200).json("Logout Succesfully");
      } else {
        res.status(400).json("you are not valid use");
      }
    } else {
      res.status(400).json("token not found");
    }
  } else {
    res.status(400).json("You are not Authenicated");
  }
});
```

Explanation of log out →

1. **Define the Route**:

```javascript
app.post("/api/logout", (req, res) => {
```

This defines a POST route at the path `/api/logout`. When a POST request is made to this path, the callback function is executed.

2. **Extract the Authorization Header**:

```javascript
const userToken = req.headers.authorization;
```

This line extracts the `authorization` header from the incoming request and assigns it to the `userToken` variable.

3. **Check if Authorization Header Exists**:

```javascript
if (userToken) {
```

This checks if `userToken` is not `null` or `undefined`. If the header is missing, the code inside this block will not run.

4. **Extract the Token from the Authorization Header**:

```javascript
const token = userToken.split(" ")[1];
```

The `authorization` header usually has the format `Bearer <token>`. This line splits the header value by a space and takes the second pa ↓ which is the actual token.

5. **Check if Token Exists**:

```javascript
if (token) {
```

This checks if the `token` extracted in the previous step is not `null` or `undefined`.

6. **Initialize Tokens Array**:

```javascript
let allTokens = [];
```

This initializes an array `allTokens` where tokens could be stored. However, since it's an empty array every time, this part of the code might not function as intended for a real-world scenario.

7. **Find the Token Index**:

```javascript
const tokenIndex = allTokens.indexOf(token);
```

This finds the index of the `token` in the `allTokens` array. If the token is not found, `indexOf` returns `-1`.

8. **Check if Token is in the Array**:

```javascript
if (tokenIndex !== -1) {
```

This checks if the token exists in the `allTokens` array (i.e., `tokenIndex` is not `-1`).

9. **Remove Token from the Array**:

```javascript
allTokens.splice(tokenIndex, 1);
res.status(200).json("Logout Succesfully");
```

If the token is found in the array, it removes the token from the array using `splice` and sends a success response.

10. **Token Not Found in Array**:

```javascript
} else {
  res.status(400).json("you are not valid use");
}
```

If the token is not found in the array, it sends a `400 Bad Request` response with a message indicating the user is not valid.

11. **Token Not Found in Header**:

```javascript
} else {
  res.status(400).json("token not found");
}
```

If the `authorization` header does not contain a token, it sends a `400 Bad Request` response with a message indicating the token was not found.

12. **Authorization Header Missing**:

```javascript
} else {
  res.status(400).json("You are not Authenticated");
}
```

If the `authorization` header is missing entirely, it sends a `400 Bad Request` response with a message indicating the user is not authenticated.

```
//creating route for logout
//after logout welcome page should open
app.get("/api/logout", (req, res) => {
  res.redirect("/");
});
app.get("/", (req, res) => {
  res.render("welcome");
});
```

Add above route after logout→

```
//craeting route for logout
//for log out we have to cut the token link so we written post
app.post("/api/logout", (req, res) => {
  const userToken = req.headers.authorization;
  if (userToken) {
    const token = userToken.split(" ")[1];
    if (token) {
      //storing tokens in array
      let allTokens = [];
      const tokenIndex = allTokens.indexOf(token);
      if (tokenIndex !== -1) {
        allTokens.splice(tokenIndex, 1);
        res.status(200).json("Logout Succesfully");
        res.redirect("/");
      } else {
        res.status(400).json("you are not valid use");
      }
    } else {
      res.status(400).json("token not found");
    }
  } else {
    res.status(400).json("You are not Authenicated");
  }
});
```

http://localhost:4000/mahesh

Hi , I am Mahesh

Logout

After clicking logout -→

http://localhost:4000

Welcome to My Application