

Fetch, Authentication and Database

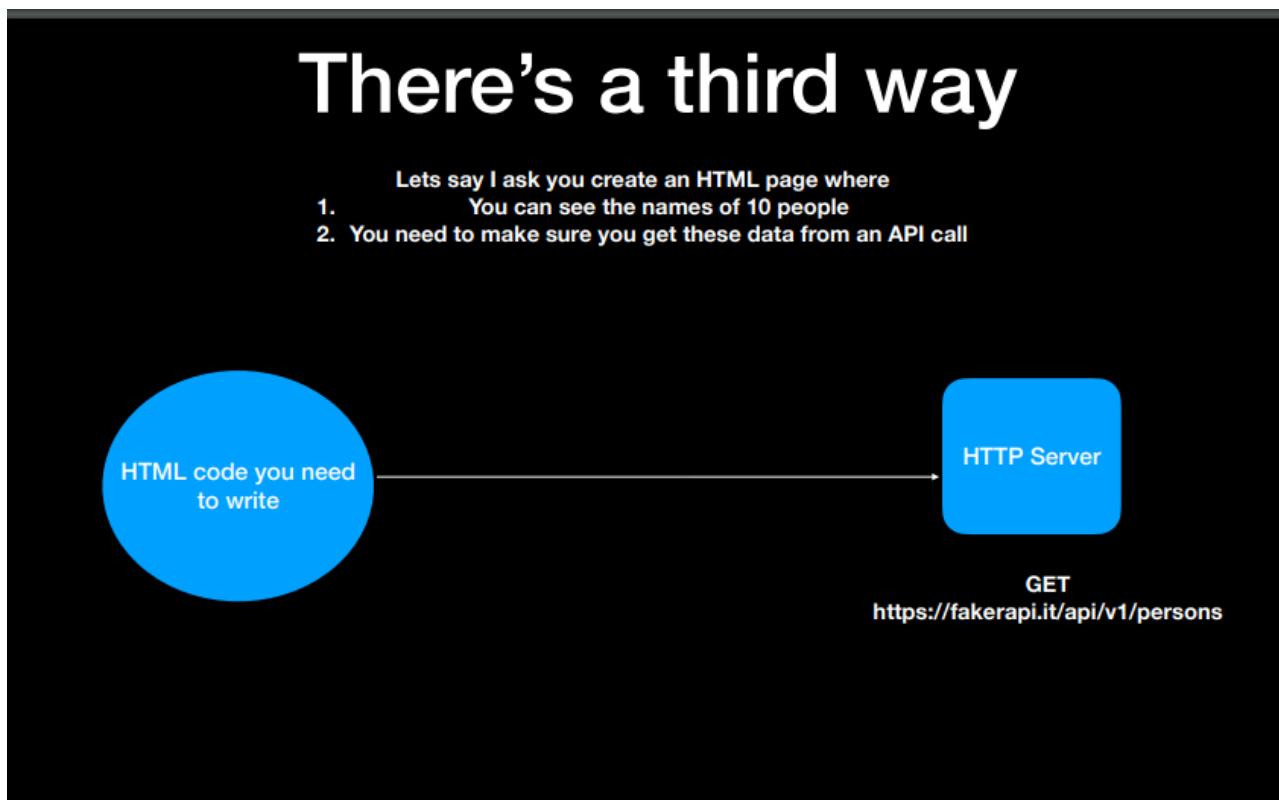
Until now we have made project which can be accessed by others easily.

The fetch API

Till now we have hit the route either by POSTMAN and browser.

How do chatgpt send request etc when a button is clicked.

Browser has a feature called fetch



Write some frontend code which hits the endpoint and get some data and put it on screen.

Simple way can be like we simply write ten name and display it.

On going into the <https://fakerapi.it/api/v1/persons> and then going on network site and then seeing the response we can see that its response contains json object contains name of 10 people.

Name	X Headers	Preview	Response	Initiator	Timing
persons			▼ {status: "OK", code: 200, total: 10,...} code: 200 ▼ data: [{id: 1, firstname: "Linwood", lastname: "Pacocha", email: "qlockman@abernathy.net",...},...] ▼ 0: {id: 1, firstname: "Linwood", lastname: "Pacocha", email: "qlockman@abernathy.net",...} ► address: {id: 0, street: "32772 Mara Cliffs Apt. 818", streetName: "Alejandra Grove", buildingNumber: "4815",...} birthday: "1937-03-27" email: "qlockman@abernathy.net" firstname: "Linwood" gender: "male" id: 1 image: "http://placeimg.com/640/480/people" lastname: "Pacocha" phone: "+5741766170489" website: "http://erdman.org" ► 1: {id: 2, firstname: "Kevin", lastname: "O'Hara", email: "thompson.kenyatta@hickle.com",...} ► 2: {id: 3, firstname: "Molly", lastname: "Stracke", email: "melvina54@collins.com",...} ► 3: {id: 4, firstname: "Gino", lastname: "Koepf", email: "franco92@reynolds.com", phone: "+4247999032039",...} ► 4: {id: 5, firstname: "Hattie", lastname: "Schmidt", email: "kozey.bryana@yahoo.com",...} ► 5: {id: 6, firstname: "Virginia", lastname: "Hintz", email: "mmante@dibbert.info",...} ► 6: {id: 7, firstname: "Wilhelmine", lastname: "Wiza", email: "amalia.rosenbaum@hotmail.com",...} ► 7: {id: 8, firstname: "Christa", lastname: "Mante", email: "garrick.wilkinson@schoen.com",...} ► 8: {id: 9, firstname: "Xavier", lastname: "Osinski", email: "theresia.okuneva@gmail.com",...} ► 9: {id: 10, firstname: "Charlene", lastname: "Upton", email: "janice.koss@franecki.com",...} status: "OK"		
favicon.ico					
			2 requests		

I need to create an HTML page that will hit this api, similar to what happens when we write a prompt in chatgpt.

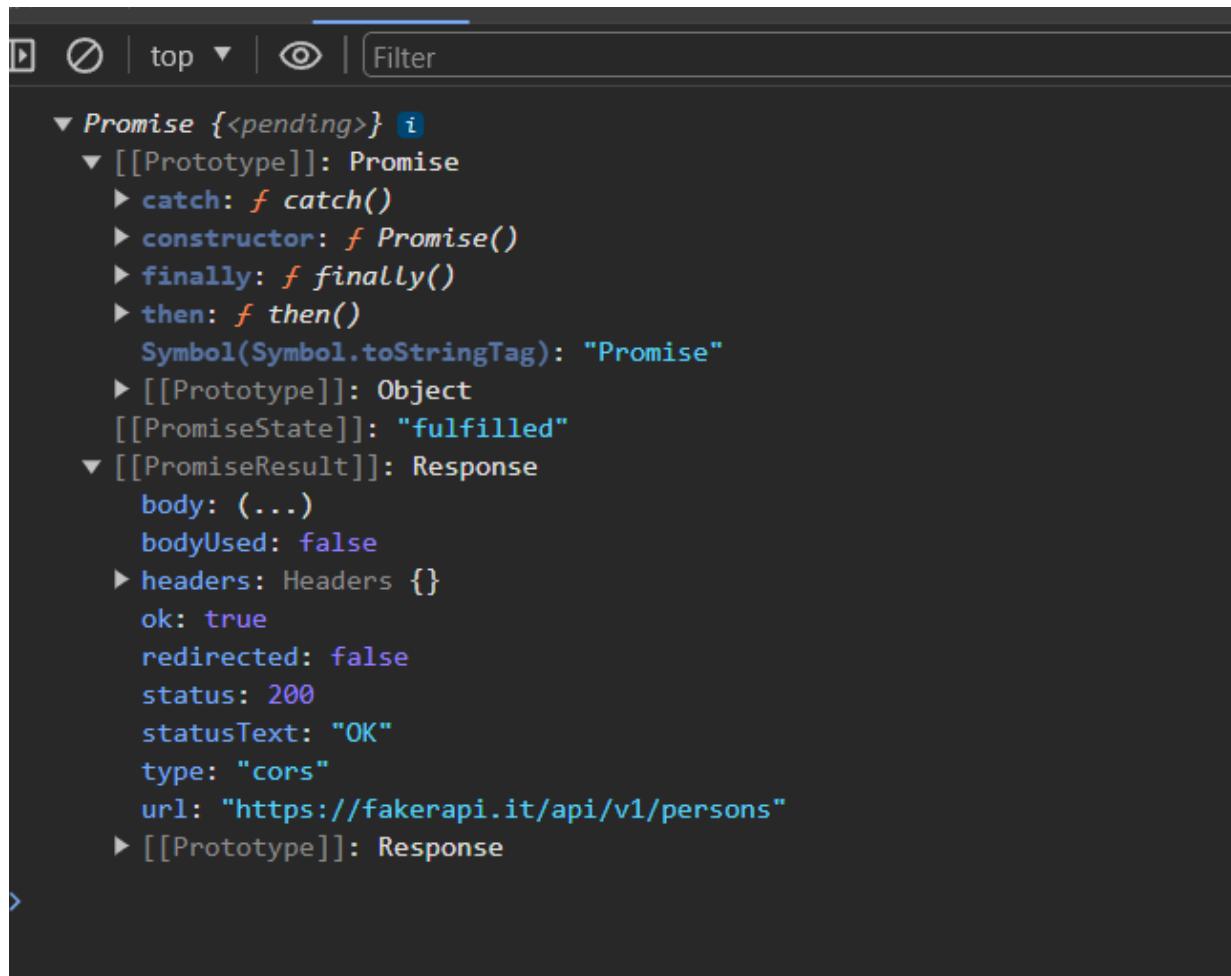
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    function getAnimalData() {
      // To check if the function is working or not
      // alert("You are inside the getAnimalData function")
      const response = fetch('https://fakerapi.it/api/v1/persons')
      //by default fetch is a get request
      // but if we want it to be a post request then we can do it
      like this
      // fetch('', {
      //   method: 'POST',
      // })
      console.log(response)
    }
  </script>
</body>
</html>
```

```

        }
    </script>
    <button onclick="getAnimalData()">
        Get Animal Data
    </button>
</body>
</html>

```

Console



It console.log() promise

So we have to use promise syntax

```

function getAnimalData() {
    fetch('https://fakerapi.it/api/v1/persons')
        .then(function(response) {
            //this also is promise

```

```

        response.json()
      .then(function(finalData) {
        console.log(finalData)
      })
    }
  }
}

```

It will console

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. It displays two objects that have been logged to the console. Both objects have the same structure: a status object with code 200, total 10, and a data array of 10 items. Each item in the data array contains an id (from 1 to 10), a first name, last name, email, and phone number. The data is displayed in a hierarchical tree view where each object has its properties expanded.

```

{
  "status": "OK",
  "code": 200,
  "total": 10,
  "data": [
    {"id": 1, "firstname": "Nigel", "lastname": "Heidenreich", "email": "hkassulke@yahoo.com", "phone": "+2744548349880", ...},
    {"id": 2, "firstname": "Valentine", "lastname": "Armstrong", "email": "mariela33@gmail.com", "phone": "+5448667044853", ...},
    {"id": 3, "firstname": "Reyna", "lastname": "Durgan", "email": "else56@hotmail.com", "phone": "+2757764392221", ...},
    {"id": 4, "firstname": "Merl", "lastname": "Frami", "email": "yvonne.mcglynn@legros.org", "phone": "+5271056559512", ...},
    {"id": 5, "firstname": "Letha", "lastname": "Heller", "email": "cspinka@hotmail.com", "phone": "+9751228803816", ...},
    {"id": 6, "firstname": "Jerald", "lastname": "Nolan", "email": "gail.doyle@yahoo.com", "phone": "+7875854164932", ...},
    {"id": 7, "firstname": "Jordon", "lastname": "Pollich", "email": "leo41@yahoo.com", "phone": "+4541818242586", ...},
    {"id": 8, "firstname": "Monroe", "lastname": "Rodriguez", "email": "hgleichner@leffler.biz", "phone": "+7270987378025", ...},
    {"id": 9, "firstname": "Magdalena", "lastname": "Konopelski", "email": "ferry.raphaelle@crooks.info", "phone": "+4889461648085", ...},
    {"id": 10, "firstname": "Garth", "lastname": "Koss", "email": "kuhn.eugene@hotmail.com", "phone": "+9419131310056", ...}
  ],
  "length": 10
}

{
  "status": "OK",
  "code": 200,
  "total": 10,
  "data": [
    {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}
  ],
  "status": "OK",
  "total": 10
}

```

This is very similar to chat gpt. Now we can say we are frontend developer we havent wrote the code at the backend but we have write the code to hit the backend server and log it .

Another way to access(get the data) is

```

async function getAnimalData() {
  const response = fetch('https://fakerapi.it/api/v1/persons')
  const finalData = await response.json()
  console.log(finalData)
}

```

To render it we need to know **DOM**.

We don't need to click the button to access the api.

We can also make it such that it refreshes.

Final code

```
<!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>
    <script>
        function getAnimalData() {
            fetch('https://fakerapi.it/api/v1/persons')
                .then(function(response) {
                    //this also is a promise
                    response.json()
                        .then(function(finalData) {
                            console.log(finalData)
                        })
                })
        }

        async function getAnimalData() {
            const response = fetch('https://fakerapi.it/api/v1/persons')
            const finalData = await response.json()
            console.log(finalData)
        }
    </script>
    <button onclick="getAnimalData()">
        Get Animal Data
    </button>
</body>
</html>
```

Authentication

Project for today –

Let people sign up to your website

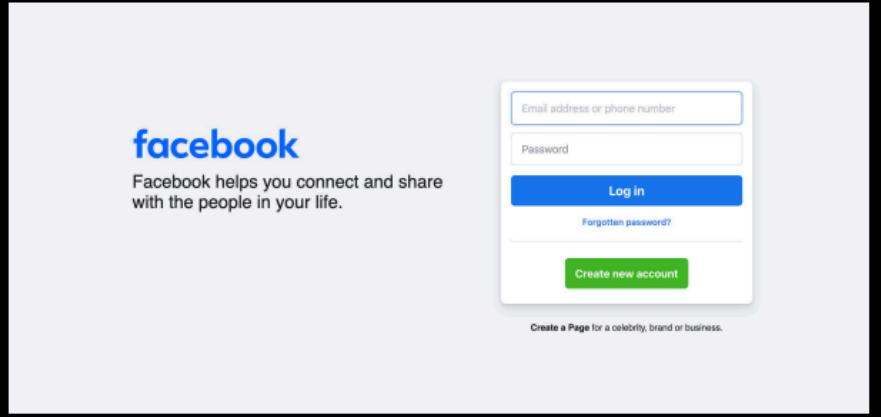
Only allow signed in users to see people (create a dummy people list)

Authentication

Almost all websites have auth

There are complicated ways
(Login with google...) to do auth

Easiest is a username password
based auth

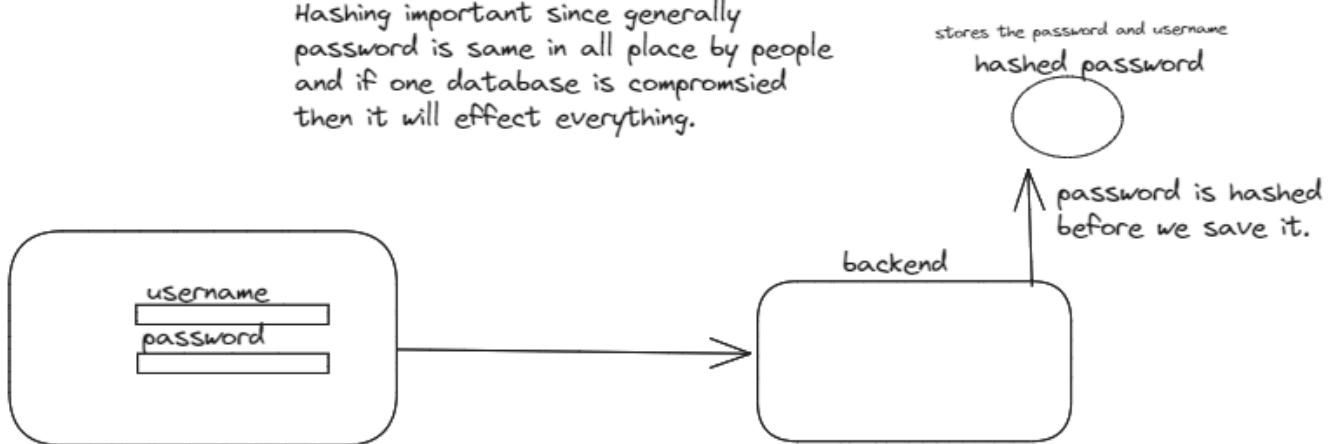


Before we get into authentication . Lets understand some cryptography jargon

1. Hashing
2. Encryption
3. Json web tokens
4. Local storage

Hashing

Hashing important since generally password is same in all place by people and if one database is compromised then it will effect everything.



whenever user re-enter username and password to conform whether password input is correct or not it will convert into gibberish and if it is same as before then its correct password.

Hashing is one way.

Hashing is one directional. Given the output no one can find out the input.

Changing the input a lil bit changes the output by lot.

Encryption:

It is two way provided we have a key.

Suppose we upload any video and photos it wont be stored directly in form of mp4 , it will be encrypted and then stored in database and when we have to fetch it we will decrypt it.

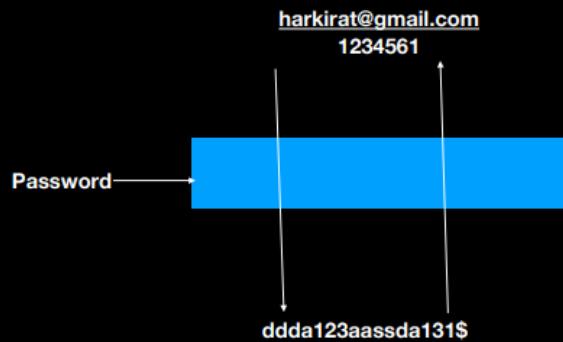
Encryption does require password

If someone has password then it can be decrypted.

Authentication

1. Hashing
2. Encryption
3. Json web tokens
4. Local storage

1. Encryption is two way
2. A string is encrypted using a password
3. String can be decrypted using the same password



Json web tokens(JWT)

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

Json : it is some hashing function but it works only for json input

It give long string it has three parts etc

Web

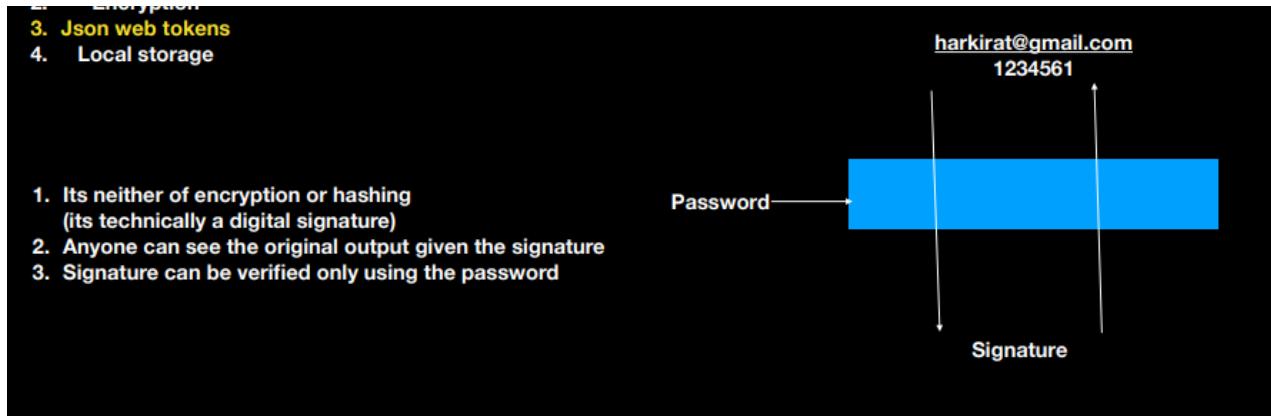
token : the long string it give is called token

We give it input and it give output and if someone has output then it can be accessed.

The very long thing which we see it in the chatgpt headers section of network. Is the JWT

We can decode JWT by going to jet.io

Why will we even do this if someone can look into this can decode it???



To decode we need password

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE	
<pre>{ "alg": "HS256", "typ": "JWT" }</pre>	
PAYOUT: DATA	
<pre>{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }</pre>	
VERIFY SIGNATURE	
HMACSHA256(

Conversion anyone can do , but verification can be done by chatgpt only.

`jwt.verify(__)`

`// this code only runs when it will be correct.`

Local Storage

We get the token when we signed in. from backend

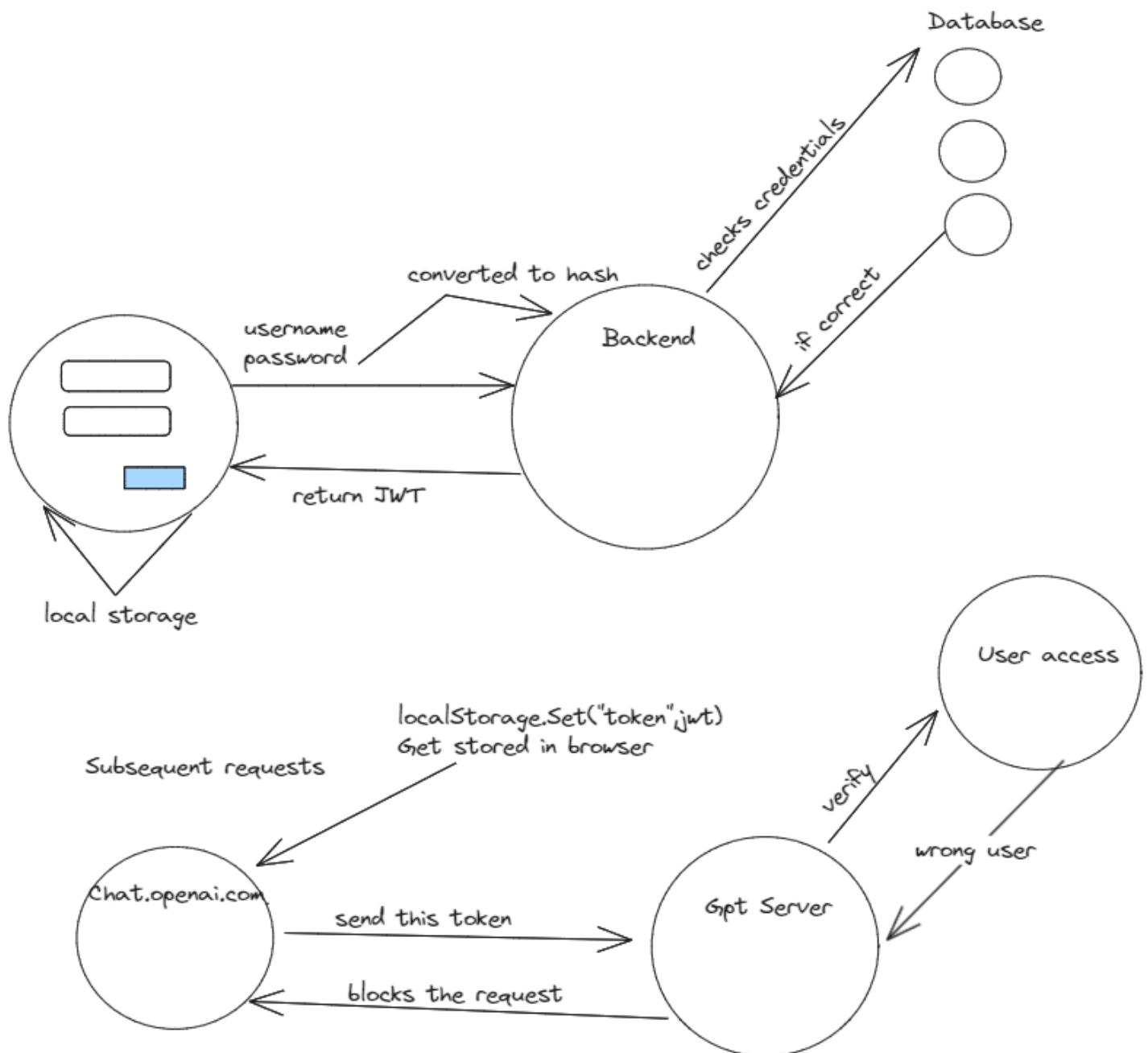
How is token getting rellaled back in every request.

Storage we can see Local storage we store inside it (machine) even if we close the browser or restart the machine the token will still be there

Frontend dev write code like this

(Any request which me make will carry this token with itself) and when i logout then they will just remove token from local storage

```
async function getAnimalData() {  
    const response = await fetch("https://fakeapri.it/conversation ",{  
        headers:{  
            "Authorization": localStorage.read("token")  
        }  
    });  
    Const finalData = await response.json()  
}
```



Q/Na

1. Why are we using JWT instead of encryption or others

It save us a database calls. (check on internet) also mode adopted way

2. Fetch recall

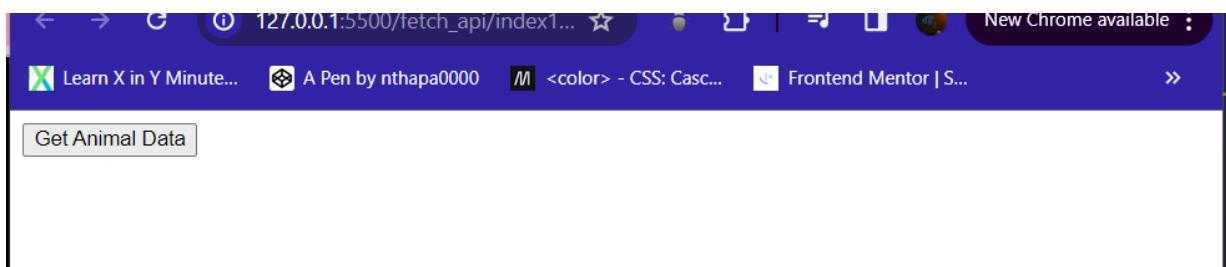
Code:

```

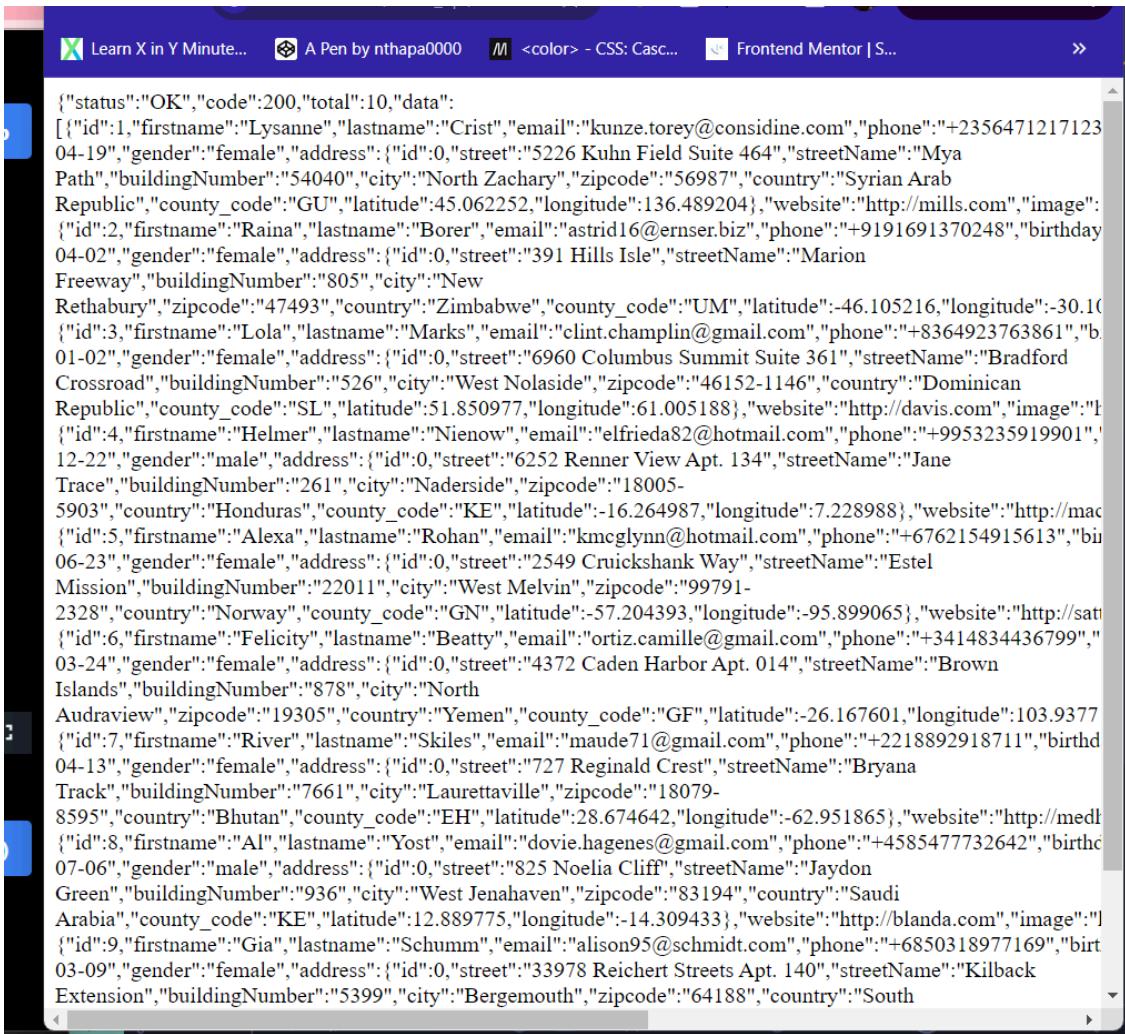
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        async function getAnimalData() {
            const response = await
fetch('https://fakerapi.it/api/v1/persons')
            const finalData = await response.json()
            document.getElementById('userData').innerHTML =
JSON.stringify(finalData)
        }
    </script>
    <div id="userData">
        <button onclick="getAnimalData()">
            Get Animal Data
        </button>
    </div>
</body>
</html>

```

Webpage



When we click on button Get Animal Data



The screenshot shows a browser window with the following tabs:

- Learn X in Y Minute...
- A Pen by nthapa0000
- <color> - CSS: Cascade
- Frontend Mentor | S...

The main content area displays a JSON array of 100 person objects. Each object has the following structure:

```
{"status": "OK", "code": 200, "total": 10, "data": [{"id": 1, "firstname": "Lysanne", "lastname": "Crist", "email": "kunze.torey@considine.com", "phone": "+235647121712304-19", "gender": "female", "address": {"id": 0, "street": "5226 Kuhn Field Suite 464", "streetName": "Mya Path", "buildingNumber": "54040", "city": "North Zachary", "zipcode": "56987", "country": "Syrian Arab Republic", "county_code": "GU", "latitude": 45.062252, "longitude": 136.489204}, "website": "http://mills.com", "image": {"id": 2, "firstname": "Raina", "lastname": "Borer", "email": "astrid16@ernser.biz", "phone": "+9191691370248", "birthday": "04-02", "gender": "female", "address": {"id": 0, "street": "391 Hills Isle", "streetName": "Marion Freeway", "buildingNumber": "805", "city": "New Rethabury", "zipcode": "47493", "country": "Zimbabwe", "county_code": "UM", "latitude": -46.105216, "longitude": -30.1001-02}, "gender": "female", "address": {"id": 0, "street": "6960 Columbus Summit Suite 361", "streetName": "Bradford Crossroad", "buildingNumber": "526", "city": "West Nolaside", "zipcode": "46152-1146", "country": "Dominican Republic", "county_code": "SL", "latitude": 51.850977, "longitude": 61.005188}, "website": "http://davis.com", "image": {"id": 4, "firstname": "Helmer", "lastname": "Nienow", "email": "elfrieda82@hotmail.com", "phone": "+9953235919901", "birthday": "12-22", "gender": "male", "address": {"id": 0, "street": "6252 Renner View Apt. 134", "streetName": "Jane Trace", "buildingNumber": "261", "city": "Naderside", "zipcode": "18005-5903", "country": "Honduras", "county_code": "KE", "latitude": -16.264987, "longitude": 7.228988}, "website": "http://mac{"id": 5, "firstname": "Alexa", "lastname": "Rohan", "email": "kmcglynn@hotmail.com", "phone": "+6762154915613", "birthday": "06-23", "gender": "female", "address": {"id": 0, "street": "2549 Cruickshank Way", "streetName": "Estel Mission", "buildingNumber": "22011", "city": "West Melvin", "zipcode": "99791-2328", "country": "Norway", "county_code": "GN", "latitude": -57.204393, "longitude": -95.899065}, "website": "http://sat{"id": 6, "firstname": "Felicity", "lastname": "Beatty", "email": "ortiz.camille@gmail.com", "phone": "+3414834436799", "birthday": "03-24", "gender": "female", "address": {"id": 0, "street": "4372 Caden Harbor Apt. 014", "streetName": "Brown Islands", "buildingNumber": "878", "city": "North Audraview", "zipcode": "19305", "country": "Yemen", "county_code": "GF", "latitude": -26.167601, "longitude": 103.9377}, "website": "http://medl{"id": 7, "firstname": "River", "lastname": "Skiles", "email": "maude71@gmail.com", "phone": "+2218892918711", "birthday": "04-13", "gender": "female", "address": {"id": 0, "street": "727 Reginald Crest", "streetName": "Bryana Track", "buildingNumber": "7661", "city": "Laurettaville", "zipcode": "18079-8595", "country": "Bhutan", "county_code": "EH", "latitude": 28.674642, "longitude": -62.951865}, "website": "http://medl{"id": 8, "firstname": "Al", "lastname": "Yost", "email": "dovie.hagenes@gmail.com", "phone": "+4585477732642", "birthday": "07-06", "gender": "male", "address": {"id": 0, "street": "825 Noelia Cliff", "streetName": "Jaydon Green", "buildingNumber": "936", "city": "West Jenahaven", "zipcode": "83194", "country": "Saudi Arabia", "county_code": "KE", "latitude": 12.889775, "longitude": -14.309433}, "website": "http://blanda.com", "image": {"id": 9, "firstname": "Gia", "lastname": "Schumm", "email": "alison95@schmidt.com", "phone": "+6850318977169", "birthday": "03-09", "gender": "female", "address": {"id": 0, "street": "33978 Reichert Streets Apt. 140", "streetName": "Kilback Extension", "buildingNumber": "5399", "city": "Bergemouth", "zipcode": "64188", "country": "South"}]}
```

3. We use Parse when we want to throw an exception and we use safeParse success without exception
4. Where to store the key in encryption-decryption
Some people store it in GitHub secret etc.
5. We can also use axios library instead of fetch.

```
const response = axios.get('https://fakeapi.it/api/v1/persons')
```

Assignment

Lets start by creating our assignment for today
A website which has 2 endpoints -

<p>POST /signin Body - { username: string password: string } Returns a json web token with username encrypted</p>	<p>GET /users Headers - Authorization header Returns an array of all users if user is signed in (token is correct) Returns 403 status code if not</p>
---	---

<https://gist.github.com/hkirat/1618d30e03dc2c276b1cd4b351028d14>

Initial codebase

```
const express = require("express");
const jwt = require("jsonwebtoken");
const jwtPassword = "123456";

const app = express();
// in memory users stored in an array
const ALL_USERS = [
  {
    username: "harkirat@gmail.com",
    password: "123",
    name: "harkirat singh",
  },
  {
    username: "raman@gmail.com",
    password: "123321",
    name: "Raman singh",
  },
  {
    username: "priya@gmail.com",
    password: "123321",
    name: "Priya kumari",
  },
];
```

```
function userExists(username, password) {
  // write logic to return true or false if this user exists
  // in ALL_USERS array
}

// exposed to to endpoints
app.post("/signin", function (req, res) {
  const username = req.body.username;
  const password = req.body.password;

  if (!userExists(username, password)) {
    return res.status(403).json({
      msg: "User doesn't exist in our in memory db",
    });
  }

  var token = jwt.sign({ username: username }, "shhhhh");
  return res.json({
    token,
  });
});

app.get("/users", function (req, res) {
  const token = req.headers.authorization;
  try {
    // verifying the token send as headers
    const decoded = jwt.verify(token, jwtPassword);
    const username = decoded.username;
    // return a list of users other than this username
  } catch (err) {
    return res.status(403).json({
      msg: "Invalid token",
    });
  }
});

app.listen(3000)
```

Solution:

<https://www.npmjs.com/package/jsonwebtoken> to know more about jsonwebtoken

npm install jsonwebtoken express

Code:

```
const express = require("express");
// import json webtokens, it is by ietf
const jwt = require("jsonwebtoken");
const jwtPassword = "123456";
// usually stored in some secret file

const app = express();
app.use(express.json());
//if no such line then it wont be able to decode the body

// in memory database
const ALL_USERS = [
  {
    username: "harkirat@gmail.com",
    password: "123",
    name: "harkirat singh",
  },
  {
    username: "raman@gmail.com",
    password: "123321",
    name: "Raman singh",
  },
  {
    username: "priya@gmail.com",
    password: "123321",
    name: "Priya kumari",
  },
];

function userExists(username, password) {
```

```

// write logic to return true or false if this user exists
// in ALL_USERS array
// todo: try to use the find function in js
let userExists = false;
for(let i=0;i<ALL_USERS.length;i++) {
    if(ALL_USERS[i].username == username && ALL_USERS[i].password == password) {
        // even if one user is found we return true
        userExists = true;
    }
}
return userExists;
// cleaner way to do is find function
}

app.post("/signin", function (req, res) {
    // when request comes to sign in
    // when user send a post request , they need to send username and
    password in body
    const username = req.body.username;
    const password = req.body.password;

    if (!userExists(username, password)) {
        return res.status(403).json({
            msg: "User doesnt exist in our in memory db",
        });
    }

    var token = jwt.sign({ username: username }, jwtPassword);
    //get back the token and send to the user
    return res.json({
        // user responsibility to store the token
        token,
    });
});

app.get("/users", function (req, res) {
    // expect user will send the authorization
    const token = req.headers.authorization;

```

```

try {
  const decoded = jwt.verify(token, jwtPassword);
  const username = decoded.username;
  // return a list of users other than this username
} catch (err) {
  // if token is invalid we send 403 status code
  return res.status(403).json({
    msg: "Invalid token",
  });
}
);

app.listen(3000)

```

Sending request,

1. Username/password is not valid

The screenshot shows the Postman interface with the following details:

- HTTP Method:** POST
- URL:** http://localhost:3000/signin
- Body (JSON):**

```

1 {
2   "username": "abracadabra@gmail.com",
3   "password": "123414"
4 }

```

- Response:** 403 Forbidden
- ResponseBody (Pretty):**

```

1 {
2   "msg": "User doesnt exist in our in memory db"
3 }

```

2. Username/password is valid

The screenshot shows the Postman interface. At the top, it says "POST" and "http://localhost:3000/signin". Below that, under "Body", there are tabs for "Pretty", "Raw", "Preview", "Visualize", and "JSON". The "JSON" tab is selected, showing the following code:

```
1 {  
2   "username": "harkirat@gmail.com",  
3   "password": "123"  
4 }
```

At the bottom, the response status is shown as "200 OK" with "5 ms" and "396 B" and a "Save Response" button.

In the "Pretty" tab of the response, the token is displayed as:

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
3       eyJ1c2VybmcFtZSI6ImhhcmtpcmF0QGdtYWlsLmNvbSIsImhlhdCI6MTcwNDY1NDI3NH0.  
4       jC5VBN436GAIky2aDbJve9W77xXRFX0Xo3B_Cy_b7fg"  
5 }
```

every time we send the request even of the same user it will send a unique token as a response

If we want to decode this jwt token we can do it in jwt.io

Encoded

```
eyJhbGciOiJIUzI1NiIs  
InR5cCI6IkpXVCJ9.eyJ  
1c2VybmFtZSI6Imhhcmt  
pcmF0QGdtYWlsLmNvbSI  
sImIhdCI6MTcwNDY1NDQ  
wOX0._qf53c6vRDMP5NL  
zuXiyNJsUtxG2k9noSj3  
n-FS9pFk
```

Decoded



Now returning the list of users in get request when we successfully verify the token send by the user in the authorization headers

Code/Final code

```
const express = require("express");  
// import json webtokens, it is by ietf  
const jwt = require("jsonwebtoken");  
const jwtPassword = "123456";  
// usually stored in some secret file
```

```
const app = express();
app.use(express.json());
//if no such line then it wont be able to decode the body

// in memory database
const ALL_USERS = [
  {
    username: "harkirat@gmail.com",
    password: "123",
    name: "harkirat singh",
  },
  {
    username: "raman@gmail.com",
    password: "123321",
    name: "Raman singh",
  },
  {
    username: "priya@gmail.com",
    password: "123321",
    name: "Priya kumari",
  },
];
function userExists(username, password) {
  // write logic to return true or false if this user exists
  // in ALL_USERS array
  // todo: try to use the find function in js
  let userExists = false;
  for(let i=0;i<ALL_USERS.length;i++) {
    if(ALL_USERS[i].username == username && ALL_USERS[i].password == password) {
      // even if one user is found we return true
      userExists = true;
    }
  }
  return userExists;
}
// cleaner way to do is find function
}
```

```
app.post("/signin", function (req, res) {
    // when request comes to sign in
    // when user send a post request , they need to send username and
password in body
    const username = req.body.username;
    const password = req.body.password;

    if (!userExists(username, password)) {
        return res.status(403).json({
            msg: "User doesn't exist in our in memory db",
        });
    }

    var token = jwt.sign({ username: username }, jwtPassword);
    //get back the token and send to the user
    return res.json({
        // user responsibility to store the token
        token,
    });
});
```

```
app.get("/users", function (req, res) {
    // expect user will send the authorization
    // token in authorization header
    const token = req.headers.authorization;
    try {
        const decoded = jwt.verify(token, jwtPassword);
        const username = decoded.username;
        // return a list of users other than this username
        res.json({
            users: ALL_USERS.filter(function(value){
                if(value.username != username){
                    return true;
                }
                else{
                    return false;
                }
            })
        })
    }
})
```

```

} catch (err) { //catching errors
  // if token is invalid we send 403 status code
  return res.status(403).json({
    msg: "Invalid token",
  });
}

app.listen(3000)

```

Sending the request via postman

In the authorization header(in GET request) we send token which we get from the POST request.

This is token for the following user.

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/signin
- Body (JSON):**

```

1 {
2   "username": "harkirat@gmail.com",
3   "password": "123"
4 }

```
- Response Headers:**
 - 200 OK
 - 122 ms
 - 396 B
 - Save Response
- Response Body (Pretty):**

```

1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VybmltZSI6ImhhcmtpcmF0QGdtYWlsLmNvbSIsImhlhdCI6MTcwNDY1NTE5Mn0.
7Zxg2DDE0K4hyinAQPxUbxPCLh2AfH_mNpAfnM41E44"
3 }

```

Now we will send this token as Authorization header in GET request

We get response of data of users other than harkirat@gmail.com

GET http://localhost:3000/users

Send

Params Auth Headers (12) Body Pre-req. Tests Settings Cookies

Key	Value
username	raman
password	pass
authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ

Body 200 OK 30 ms 392 B Save Response

Pretty Raw Preview Visualize JSON

```
1 "users": [
2   {
3     "username": "raman@gmail.com",
4     "password": "123321",
5     "name": "Raman singh"
6   },
7   {
8     "username": "priya@gmail.com",
9     "password": "123321",
10    "name": "Priya kumari"
11  }
12 ]
13 ]
14 ]
```

Now if we send an invalid token or no token at all

GET http://localhost:3000/users

Send

Params Auth Headers (12) Body Pre-req. Tests Settings Cookies

Key	Value
authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC

Body 403 Forbidden 15 ms 265 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "Invalid token"
3 }
```

We will send This error message even if the token is correct but **jwtPassword** is changed from when the token was generated.

We made a real-world backend with a signin endpoint and another endpoint that does something. It is authenticated and nobody can access it unless they send correct jwt

Authentication Recap

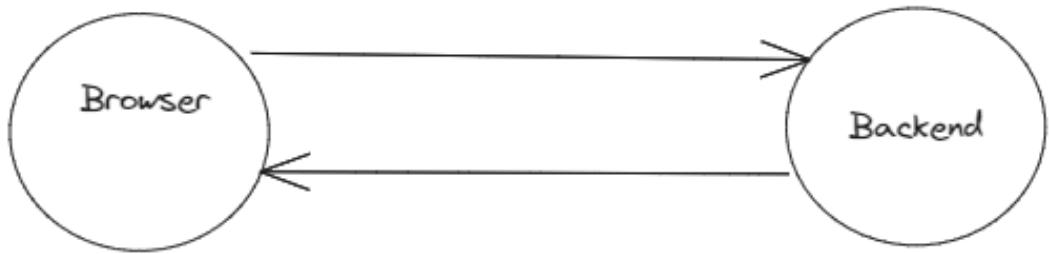
JWT to create tokens

User gets back a token after the signin request
User sends back tokens in all authenticated requests

Databases

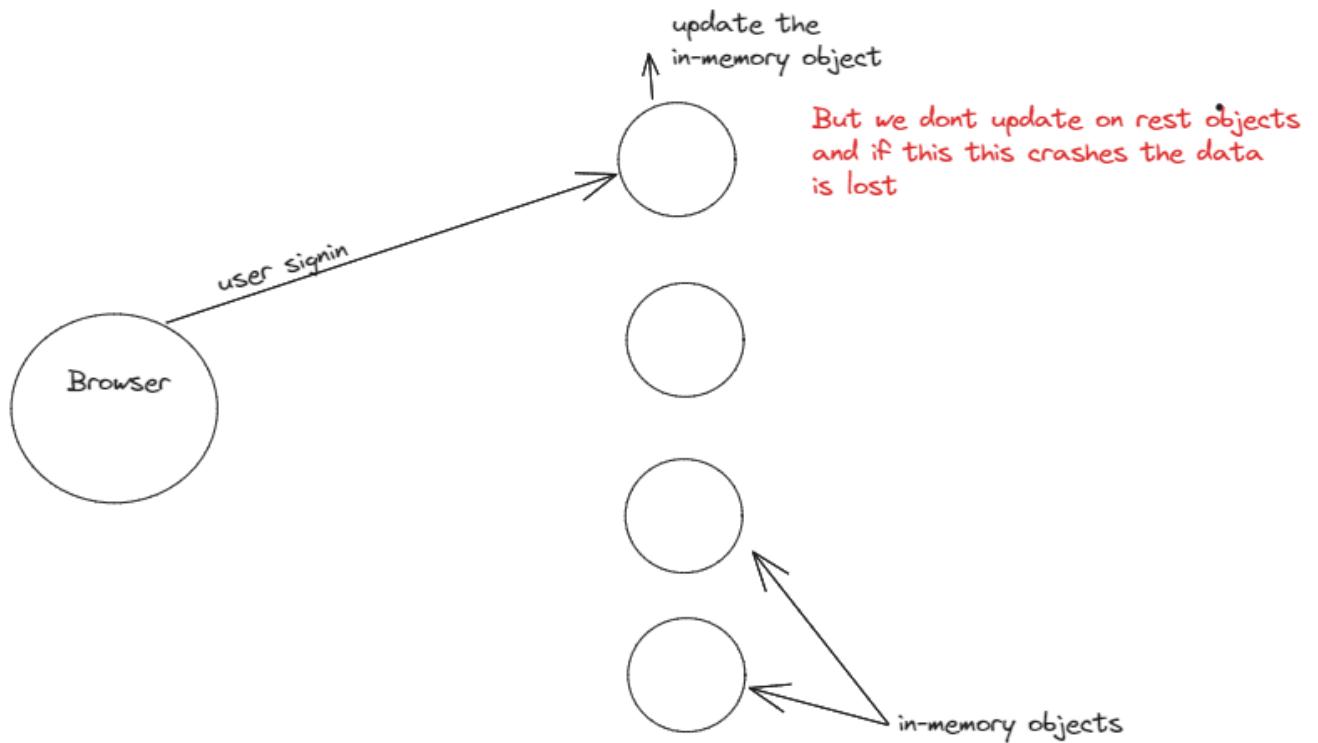
We had in-memory object

- We didnt allow users to sign in themselves
- We don't allow users to delete or leave the platform
- If the user is in the list then it fine otherwise we can't do anything
- All the updates are lost if the program crashes
- We dont have single backend server in real-world



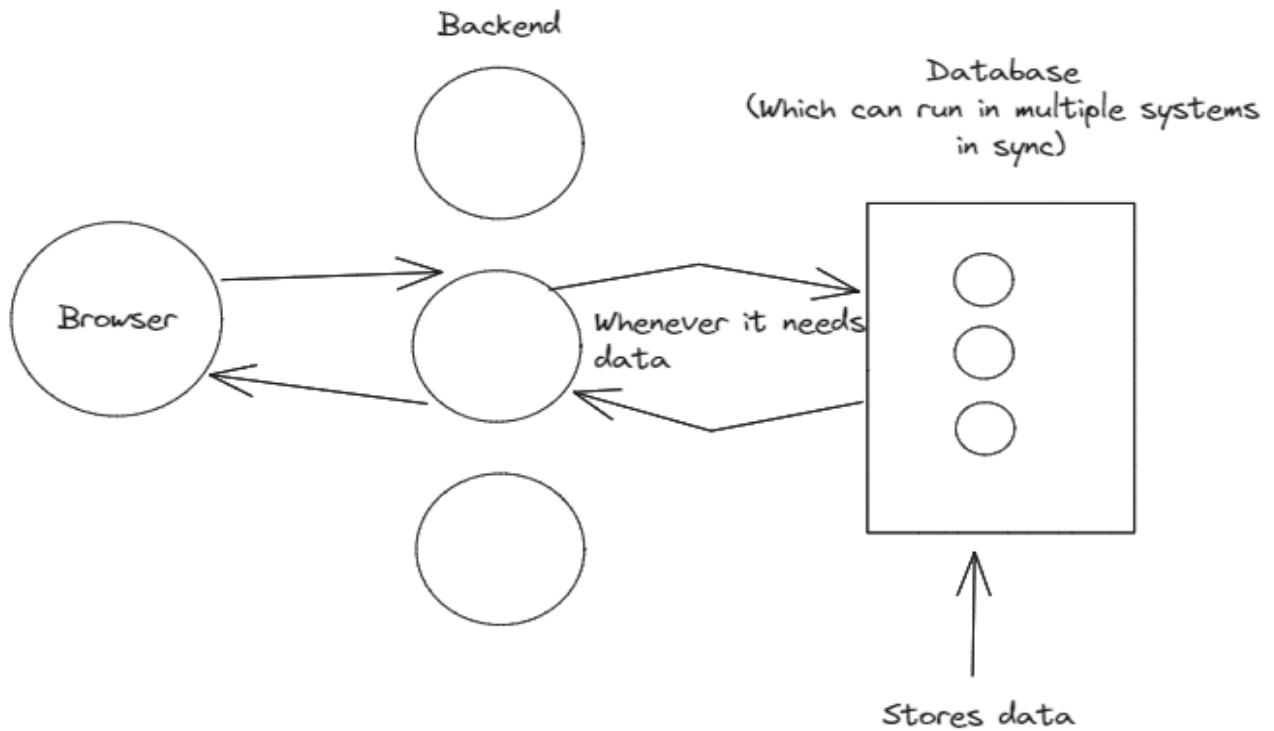
- Instead we have multiple backend servers

Why we must NOT have in-memory database in real world applications



- What will the system kinda look like

User doesn't have access to the database



How is database exposed?

How does it allow backend server to put data on it?

How does it let backend server get the data?

How does backend know the credentials of database?

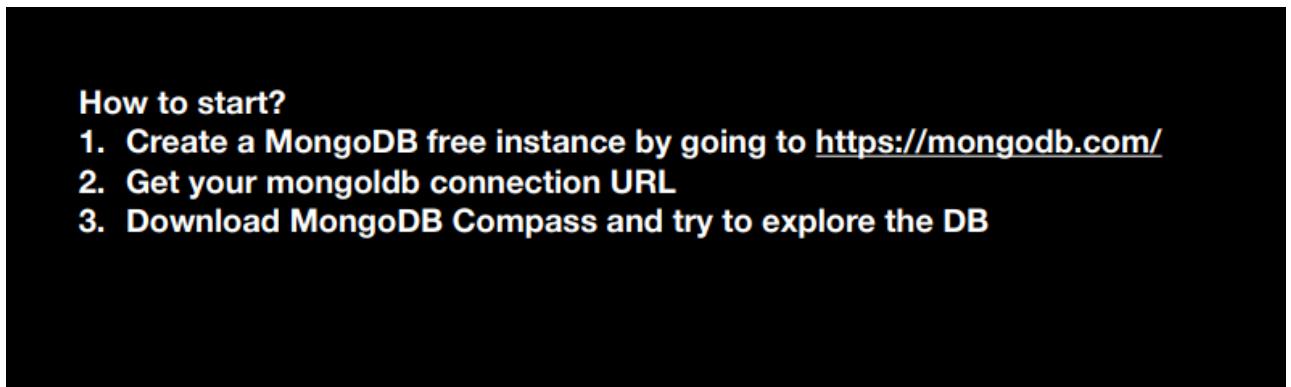
There are various type of databases

1. Graph DBs
2. Vector DBs -> ML
3. SQL DBs -> Most full stack application
4. NoSql DBsm -> MongoDB

MongoDB

- MongoDB lets you create databases (Multiple database) (Facebook has messenger ,FB , insta)

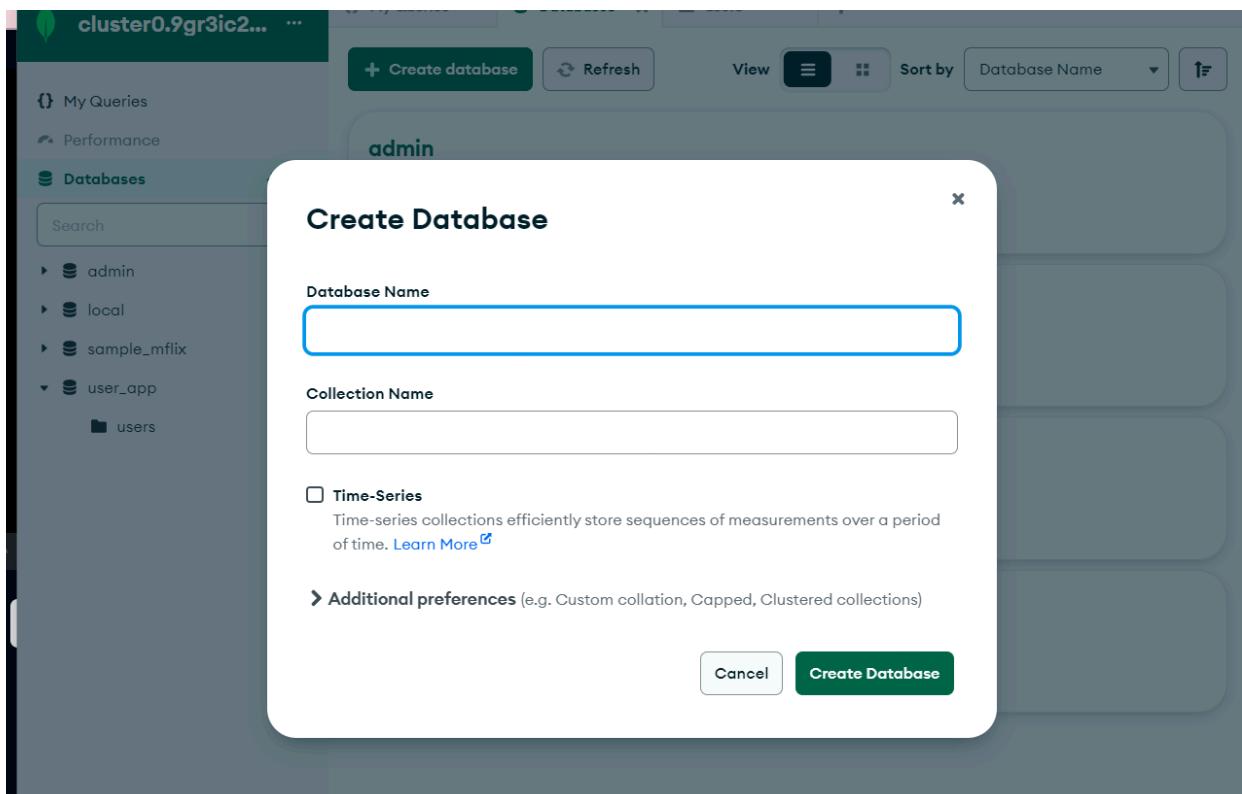
- In each DB it lets us to create tables(collections)
- In each table , it lets you dump JSON data
- It is schemaless
- It scales well and is a decent choice for most use case



Check Video 3.0.7

MongoDB compass is GUI very helpful while debugging.

Creating a database



The place where we should store the in-memory data is here.

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Import Data

Now we will create a sign up end point, and when a user sign up its data will be stored in the database.

How can the nodejs process connect to this and put data on it?

By doing this we will know how to write to a database and how to read from the database. This is what is full stack is.

How does the backend connect to the database? Using libraries!

1. Express lets u create an HTTP server
2. Jsonwebtoken library lets you create jets
3. Mongoose lets you connect to your database

Let's explore **mongoose**.

<https://mongoosejs.com/>

Let's do an Assignment:

We have to create a backend logic for the server that is connected to the database

End user can send one of three requests:

1. /signup
2. /signin (does this user exist in database and is the password correct) (we return them the jwt so they dont have to sign in again and again)
3. /users (we expect headers (jwt) and return the user the list of users)

Starting Code:

```
const express = require("express");
const jwt = require("jsonwebtoken");
const mongoose = require("mongoose");
const jwtPassword = "123456";

mongoose.connect(
  "your_mongo_url",
);

const User = mongoose.model("User", {
  name: String,
  username: String,
  password: String,
});

const app = express();
app.use(express.json());

function userExists(username, password) {
  // should check in the database
}

app.post("/signin", async function (req, res) {
```

```

const username = req.body.username;
const password = req.body.password;

if (!userExists(username, password)) {
    return res.status(403).json({
        msg: "User doesn't exist in our in memory db",
    });
}

var token = jwt.sign({ username: username }, "shhhhh");
return res.json({
    token,
}) ;
});

app.get("/users", function (req, res) {
    const token = req.headers.authorization;
    try {
        const decoded = jwt.verify(token, jwtPassword);
        const username = decoded.username;
        // return a list of users other than this username from the database
    } catch (err) {
        return res.status(403).json({
            msg: "Invalid token",
        });
    }
});

app.listen(3000);

```

Since MongoDB is schemaless it can have anything. So we use mongoose.

JavaScript

```

const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1:27017/test');

```

mongoose will first ask what are you planning to put in the database

From client side but still we can put random from mongodb

```
JavaScript
```

```
const Cat = mongoose.model('Cat', { name: String });
```

And then only allow to save (put data in database)

```
JavaScript
```

```
const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

Example:

```
const mongoose = require("mongoose");

mongoose.connect("mongodb+srv://admin:KmCihXn011podXRj@cluster0.9gr3ic2.mongodb.net/")
// put your connect here

// Describing the model / schema/ table
const User = mongoose.model('Users', { name: String, email: String,
password: String });

// user table
const user = new User({
    name: "ThapaJi",
    email: "abc@gmail.com",
    password: "123456"
});

// putting data into database
```

```
user.save();
```

npm install mongoose

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases: admin, local, sample_mflix, test, users (which is selected and highlighted in green), and user_app. The main area shows the 'test' database with the 'users' collection. The 'Documents' tab is active. There are two documents listed:

- Document 1:

```
_id: ObjectId('659c4e9b62bb8657833bb056')
name: "Thapa"
email: "abc@gmail.com"
password: "123456"
__v: 0
```
- Document 2:

```
_id: ObjectId('659c4f18f615983067ed034d')
name: "ThapaJi"
email: "abc@gmail.com"
password: "123456"
__v: 0
```

Now sending data through an http server

Code:

```
const express = require("express");
const mongoose = require("mongoose");
const app = express();

app.use(express.json());
mongoose.connect("mongodb+srv://admin:KmCihXn0llpodXRj@cluster0.9gr3ic2.mongodb.net/")

// Describing the model / schema/ table
const User = mongoose.model('Users', { name: String, email: String,
password: String });
```

```
// creating a signup endpoint which doesn't put data in in-memory database
// but in the actual database.

app.post('/signup', function(req, res) {
    const username = req.body.username;
    const password = req.body.password;
    const name = req.body.name;

    // a bug and there are many more
    // we should also check that does there exist an user with this
username

    const existingUser = await User.findOne({email: username});
    // mongoose documentation
    //CRUD: create read update delete

    if(existingUser){
        return res.status(400).json({
            msg: "User already exists"
        })
    }

    const user = new User({
        name: name,
        email: username,
        password: password
    });
    await User.create({
        //      name: name,
        //      email: username,
        //      password: password
        // })
    user.save();
    return res.json({
        msg: "User created successfully"
    })
}

app.listen(3000);
// putting data into the database
```

This code not correct yet.

Q/Na

1. What is findOne function?

It is similar to what we do in MongoDB Compass

test.users

2 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Filter { name : "Thapa" } Generate query Explain Reset Find Options

+ ADD DATA Export DATA

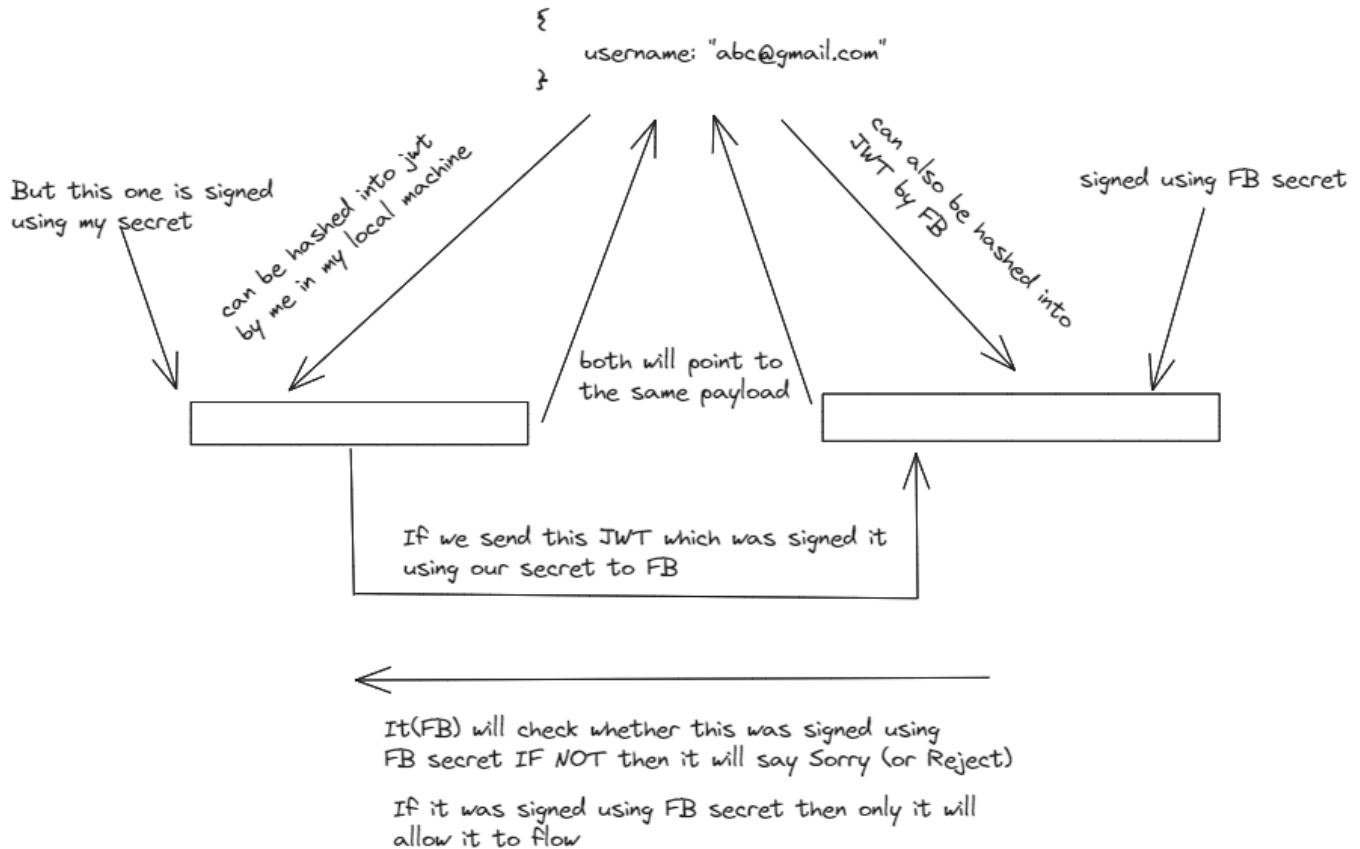
1–1 of 1

```
_id: ObjectId('659c4e9b62bb8657833bb056')
name: "Thapa"
email: "abc@gmail.com"
password: "123456"
__v: 0
```

2. User.delete and user.update
3. MongoDB and mongoose what is the difference, in the MongoDB library
we don't define schema
4. We will eventually move to axios
5. Jwt is never stored in the server, once we have Jwt, it just checks it
6. Picture like fb are stored in **object stores**

Eg Amazon S3 is commonly used for storing various media files, including pictures and videos. Its object storage system is well-suited for handling multimedia content due to several reasons.

7. Goal of jwtPassword ?? IF it can be decoded by anyone



Hence JWT password aim is to basically verify .

8. jwt.sign : will give different output mostly due to iat which it includes in payload
9. token can be of various format

By writing Bearer it tell the format of token which is coming.

Then we will have actual token

10. Why to store in database when we can store data in a file??\

- You dont have standard way to store data
- Hard to distribute
- Even if we store in mongodb (it is stored in multiple location)
- DBs are optimized for read and write
- DBs have logic for indexing to make some queries faster

11. How are Hashing and Encryption are implemented on the code level??

There are some famous algorithms for hashing and encryption (AES-128, SHA-256)

cryptojs library in node will allow us to use this algorithm

```
crypto.input("My string to encrypt").secret("AscdAfve")
```

12. If we every want to decode our data then we will use encryption and if we dont want to decode our data then we will use hashing example: password we dont want anyone to decrypt hence we use hashing and for videos we use encryption before sending it to object store.

13. Local storage function in the browser.

```
> localStorage.setItem("abc", "Hello welcome")
< undefined
> local
✖ > Uncaught ReferenceError: local is not defined
      at <anonymous>:1:1
VM12725:1
> localStorage.getItem("abc")
< 'Hello welcome'
> |
```

14. How is JWT verifying the user like what it compare with and how it verifies the signature?

```
const jwt = require("jsonwebtoken");
const password = "123456";

const token = jwt.sign({
  name: "harkirat",
},password)
// this token was created/ hashed using this password
// whoever has token can see that it is signed jwt token
// we can verify it whether this token was verified using this password
or not

// it is similar to we have cheque book in our house suppose it is issued
by the name of my father , i can see the cheque book and if i and my
father both signed the two cheque separately and submitted to the bank.
// Only one signed by my father will be accepted in the bank. The bank
verifies whether the cheque is signed by the person who was issued the
chequebook or not.
```

```
console.log(token);
```

15. Can't we run MongoDB on a local server??

Yes, we can check out the docker video of Harikat on youtube.

16. How can we store jwt token .

We store them in cookie and local storage and explicitly send them in headers

Cookie are more secure.

It is quite hard to do cookie in mobile app and hence they send fetch request along with the authorization header

```
fetch ('acs.com', {
  headers: {
    authorization: "Bearer " + localStorage.getItem("token")
  }
})
```

17. We dont need jwtPassword to decode but we need it to sign and verify.

18. Adding expiry to the jwt

```
jwt.sign({
  "name": "harkirat",
  expiresIn: new Date().getTime() + 60*60
})
```

19. jwt.decode() will take input and decode without jwt password

20. We can make middleware to check whether user exists or not.

21. Jwt requires a password during creating a token and during verification

22. findOne() function of mongoose is async since we are asking database to find something