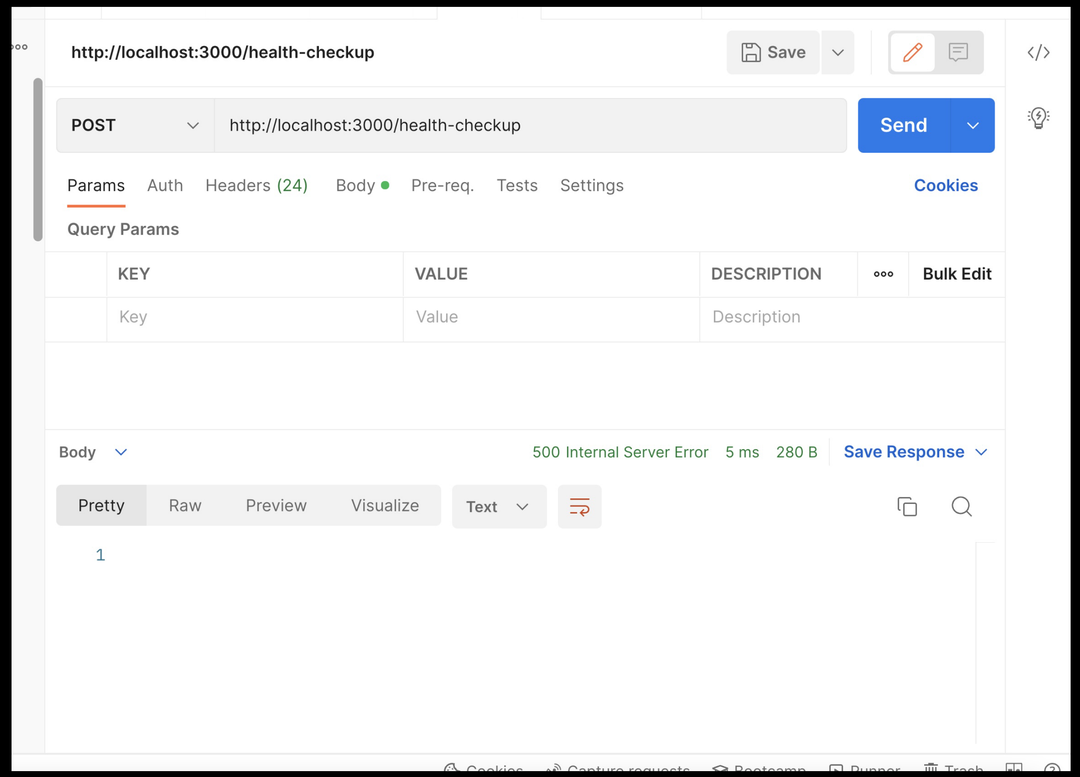
3.1
Fetch, Authentication and Databases

The fetch API

Until now, we've sent requests in 2 ways

Postman



Browser URL bar

There’s a third way

Lets say I ask you create an HTML page where

1. You can see the names of 10 people
2. You need to make sure you get these data from an API call

There’s a third way

Lets say I ask you create an HTML page where

1. **You can see the names of 10 people**
2. **You need to make sure you get these data from an API call**

index.html

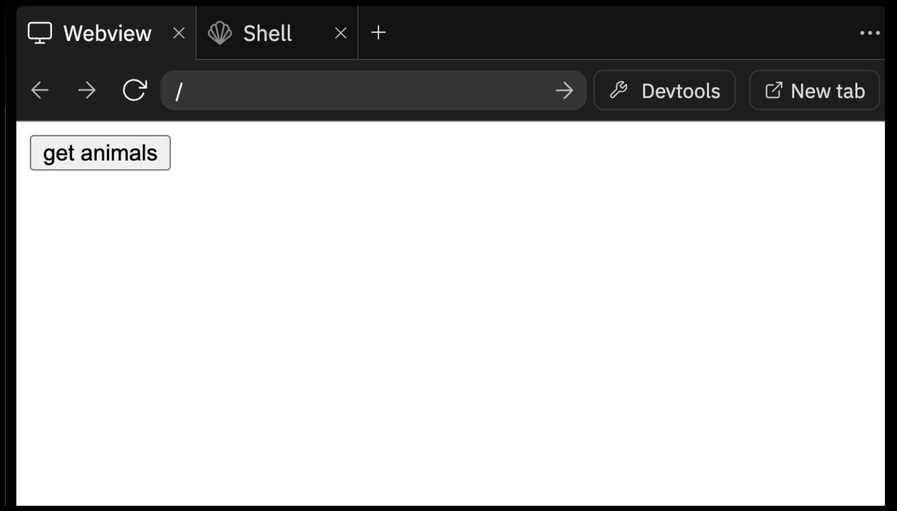
```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>replit</title>
  <link href="style.css" rel="stylesheet" type="text/css" />
</head>

<body>
  <div id="container">
  </div>
  <button onclick="getAnimals()">get animals</button>
  <script>
    function getAnimals() {
      fetch("https://fakerapi.it/api/v1/persons")
        .then(async function(response) {
          const jsonData = await response.json();
          document.getElementById("container").innerHTML = JSON.stringify(jsonData.data);
        })
    }

  </script>
</body>

</html>
```



https://gist.github.com/hkirat/ea4d132f70f69d1d47baac9eb3cc1313

Authentication

Project for today -
Let people sign up to your website
Only allow signed in users to see people (create a dummy people list)

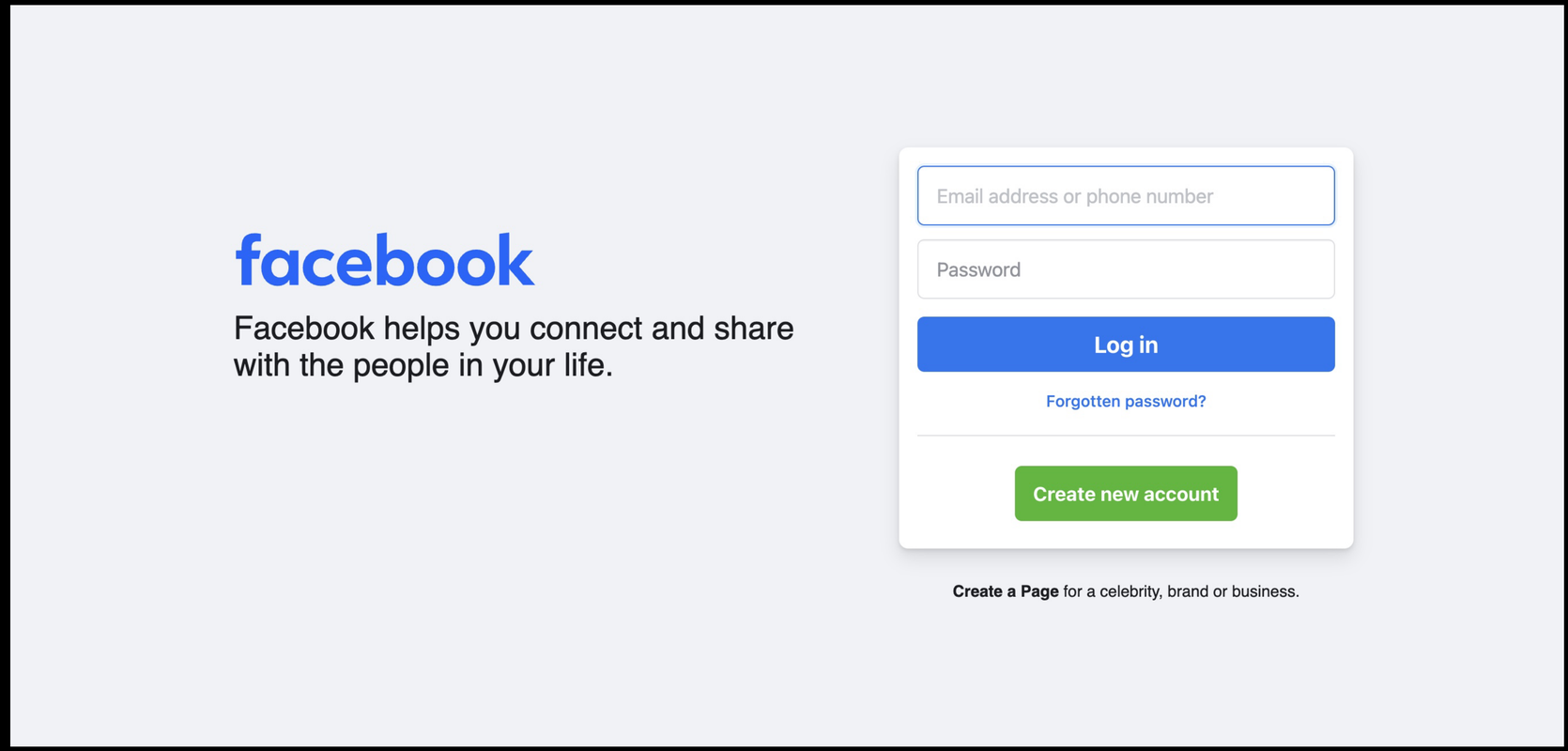
Before that, lets see
authentication

Authentication

Almost all websites have auth

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

Easiest is a username password based auth



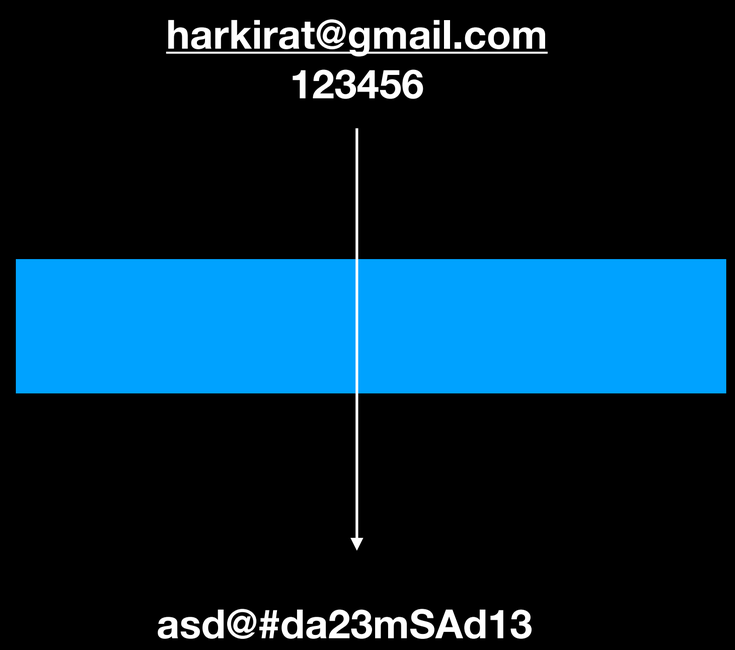
Authentication

Before we get into authentication
Lets understand some cryptography jargon

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

Authentication

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



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

Authentication

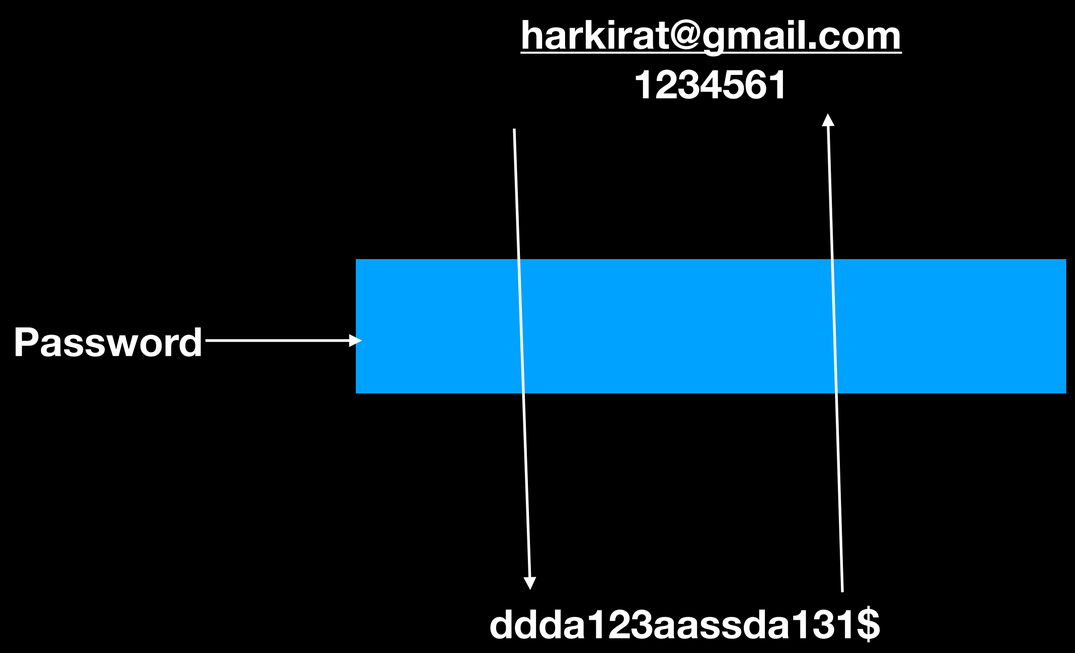
1. Hashing
2. Encryption
3. Json web tokens
4. Local storage

1. Hashing is one way
2. Given the output, no one can find out the input
3. Changing the input a lil bit changes the output by a lot

Authentication

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

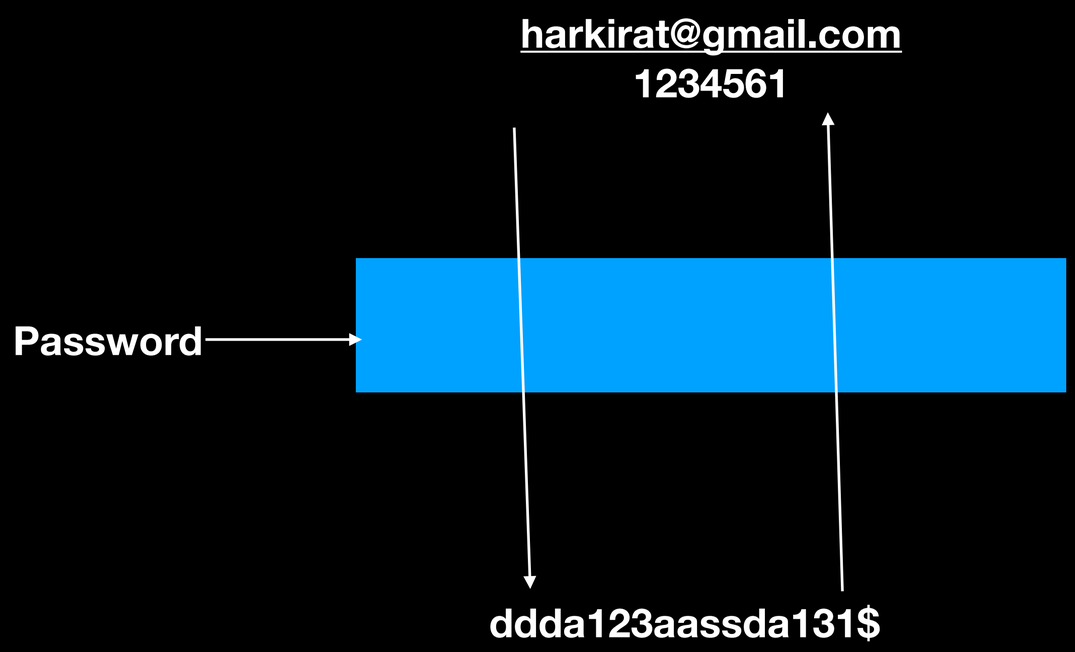
1. Encryption is two way
2. A string is encrypted using a password
3. String can be decrypted using the same password



Authentication

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

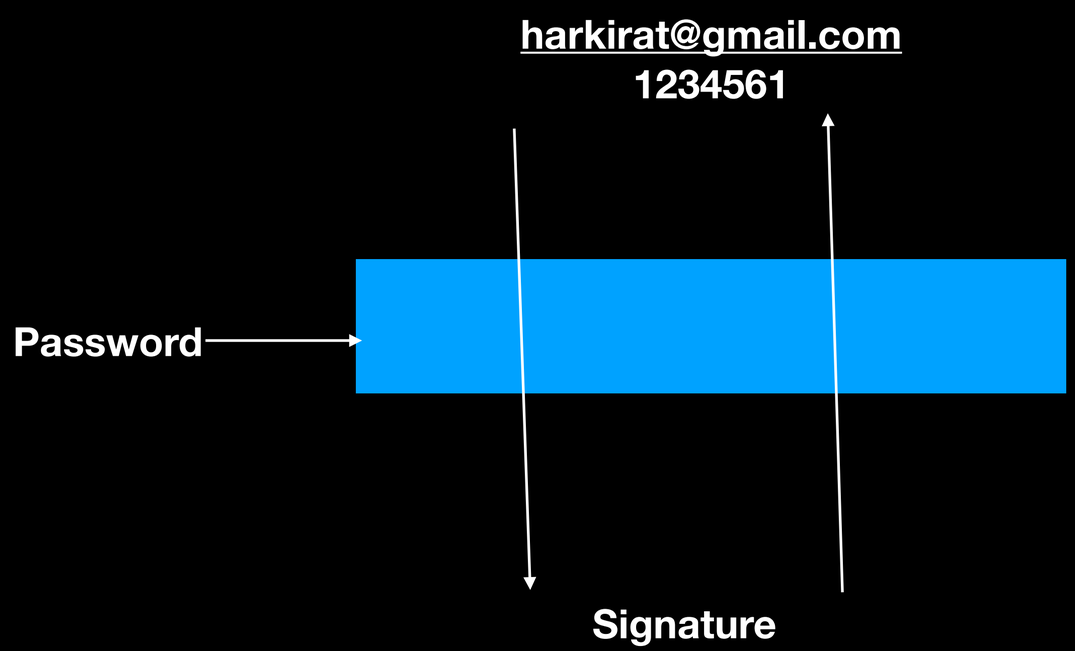
1. Encryption is two way
2. A string is encrypted using a password
3. String can be decrypted using the same password



Authentication

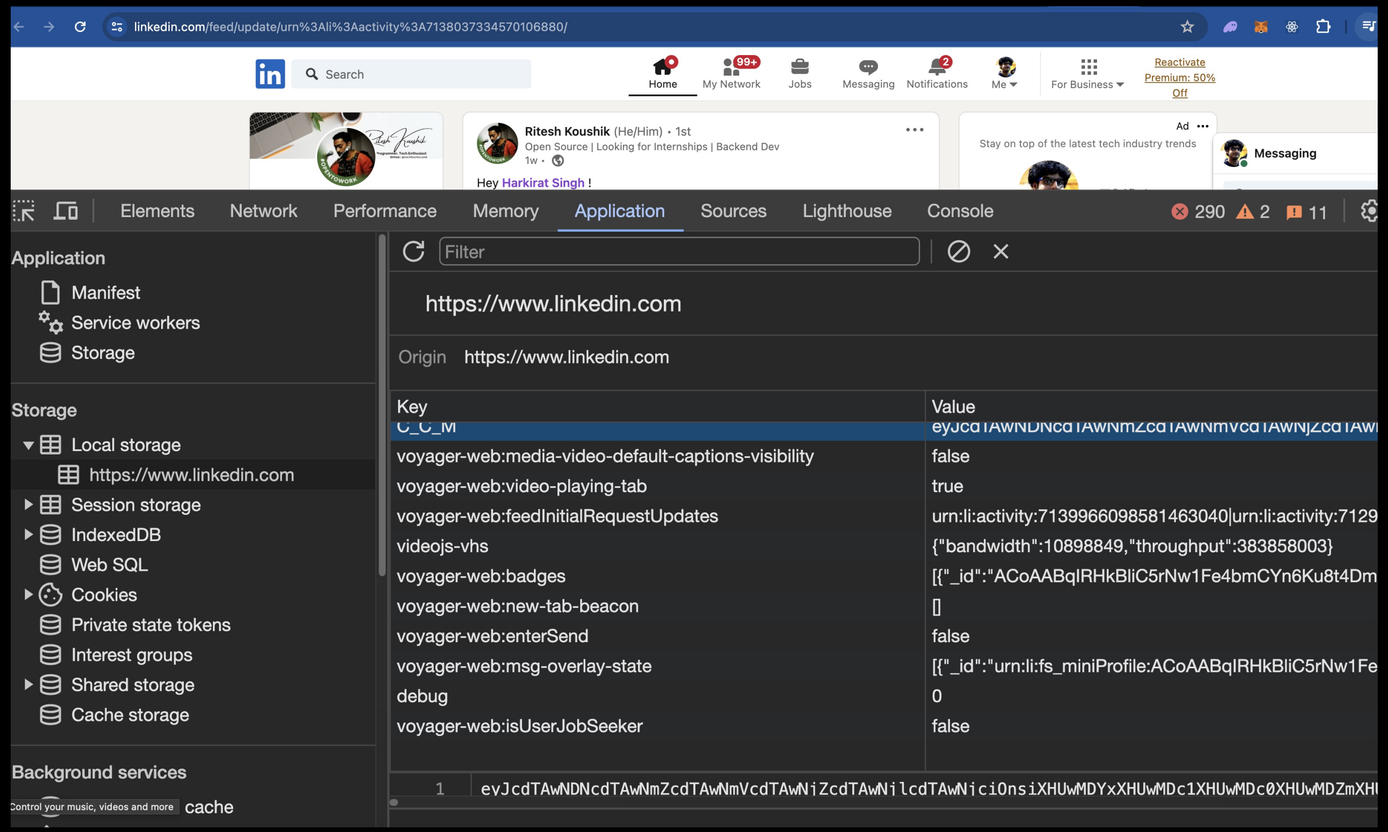
1. Hashing
2. Encryption
3. Json web tokens
4. Local storage

1. Its neither of encryption or hashing
   (its technically a digital signature)
2. Anyone can see the original output given the signature
3. Signature can be verified only using the password



Authentication

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



A place in your browser where you can store some data
Usually things that are stored include -

1. Authentication tokens
2. User language preference
3. User theme preference

Authentication

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

POST /signin
Body - {
username: string
password: string
}

Returns a json web token with username encrypted

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

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

Authentication Recap

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

Databases

Until now, we've been storing data in memory
This is bad for a few reasons -

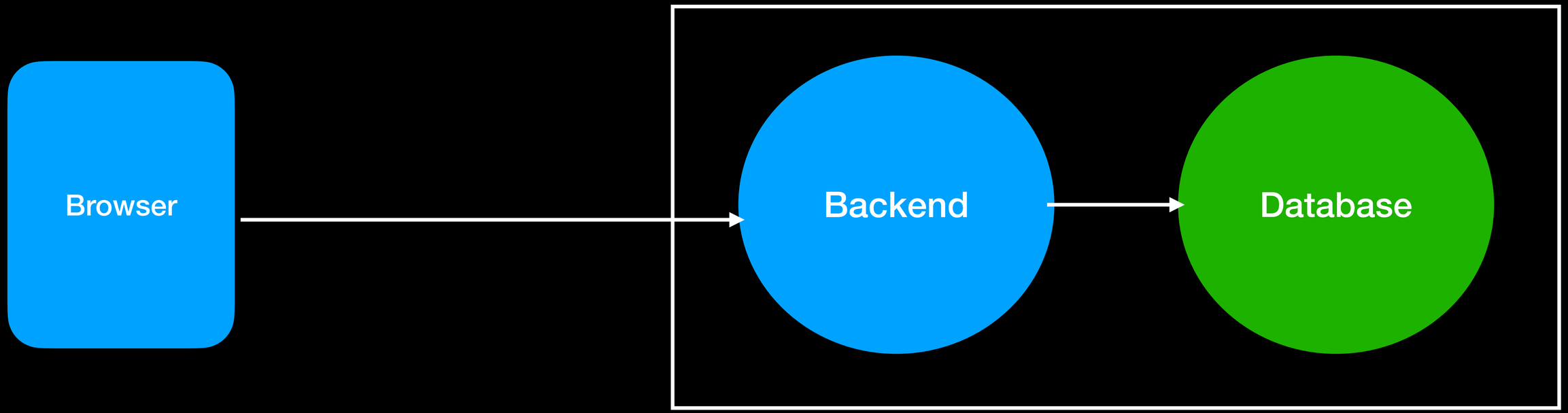
1. Data can't be dynamic, if you update in memory objects, the updates are lost if the process restarts
2. There are multiple servers in the real world

```
6
7  const ALL_USERS = [
8    {
9      username: "harkirat@gmail.com",
10     password: "123",
11     name: "harkirat singh",
12   },
13   {
14     username: "raman@gmail.com",
15     password: "123321",
16     name: "Raman singh",
17   },
18   {
19     username: "priya@gmail.com",
20     password: "123321",
21     name: "Priya kumari",
22   },
23 ];
24
```

Databases

In the real world, a basic architecture looks like this

User hits the backend
Backend hits the database
User doesn’t have access to the database/can’t talk to the DB



Databases

In the real world, a basic architecture looks like this

There are various types of databases

1. Graph DBs
2. Vector DBs
3. SQL DBs
4. NoSql DBs

For todays class, we'll look at a famous NoSQL database - MongoDb

Databases

MongoDB lets you create databases
In each DB, it lets you 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 cases

Databases

How to start?

1. **Create a MongoDB free instance by going to https://mongodb.com/**
2. **Get your mongoldb connection URL**
3. **Download MongoDB Compass and try to explore the DB**

Databases

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

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

Databases

Lets explore mongoose and do the next assignment
https://mongoosejs.com/
https://gist.github.com/hkirat/23c42247d8a37de53b005d2668507a67