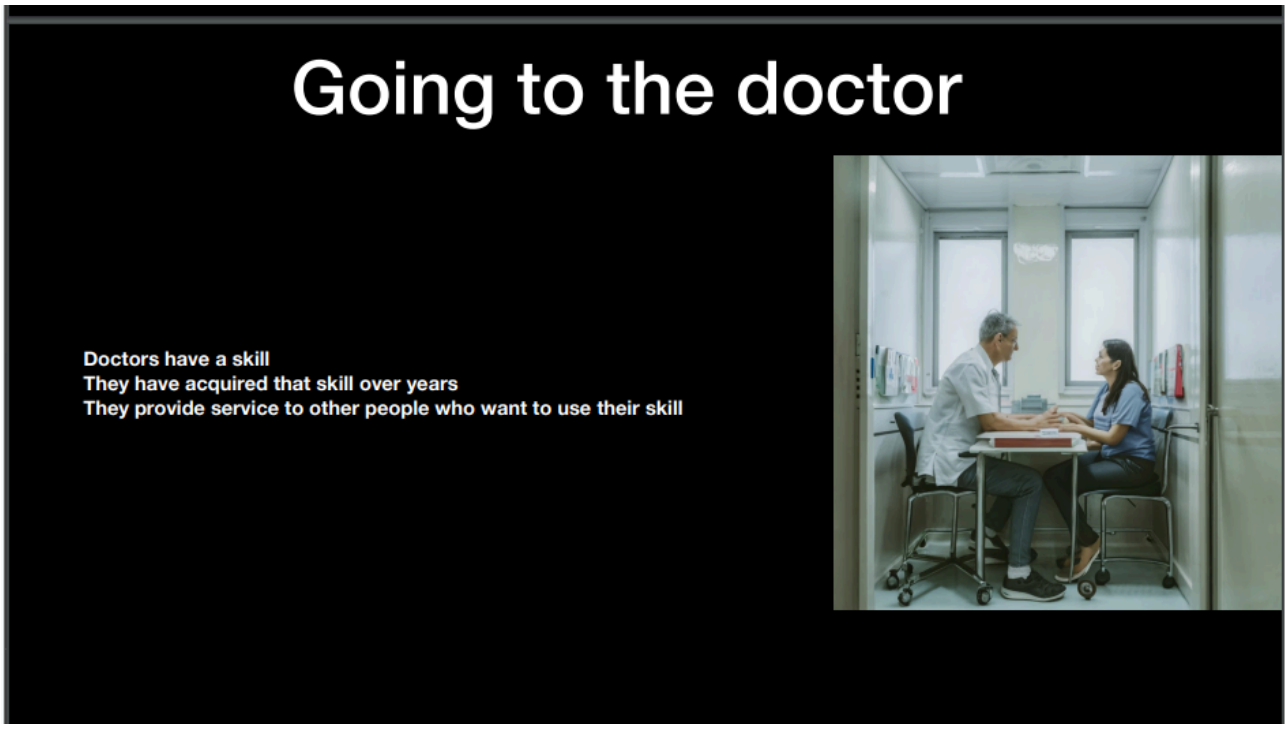
Express With Examples

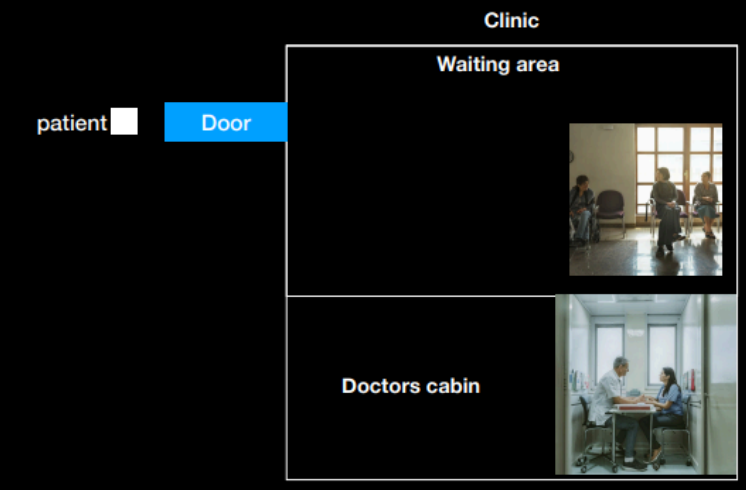
We are taking the example of a Doctor.



Going to the doctor

To expose this life skill, they open a clinic
People who want to use their skill line up in a waiting room
One by one, the doctor meets with them
The doctor is single threaded

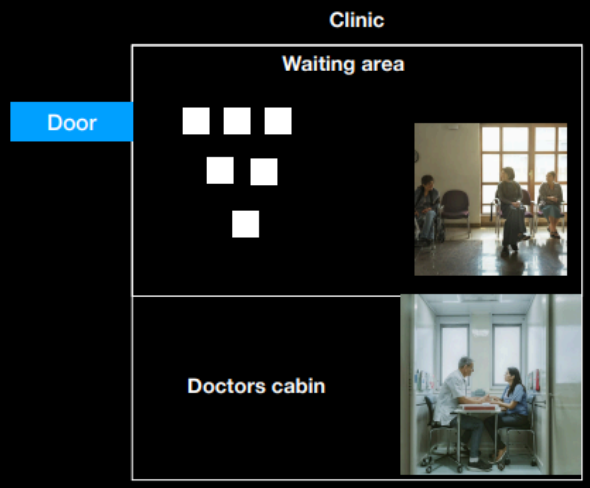
Going to the doctor



How do people reach the doctors?
They get their address and navigate to it

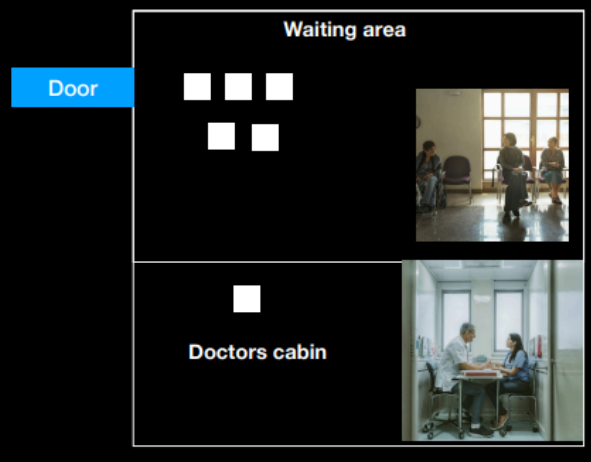
Going to the doctor

Once they reach there, they wait in the waiting area
Until their time comes



Going to the doctor

Doctor tends to them one by one



Going to the doctor

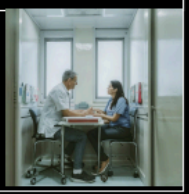
Clinic

Waiting area

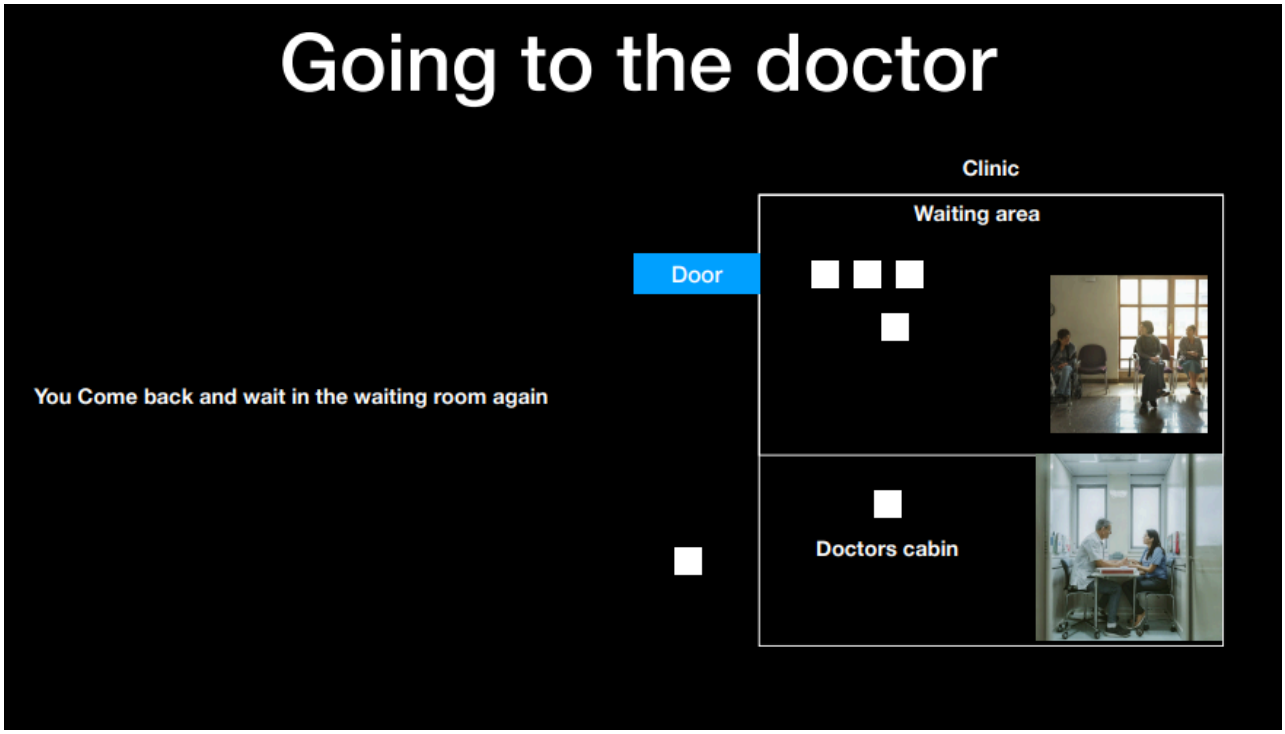
Door

Doctor can tell them to get a medicine in the middle and meanwhile tend to other people

Doctors cabin

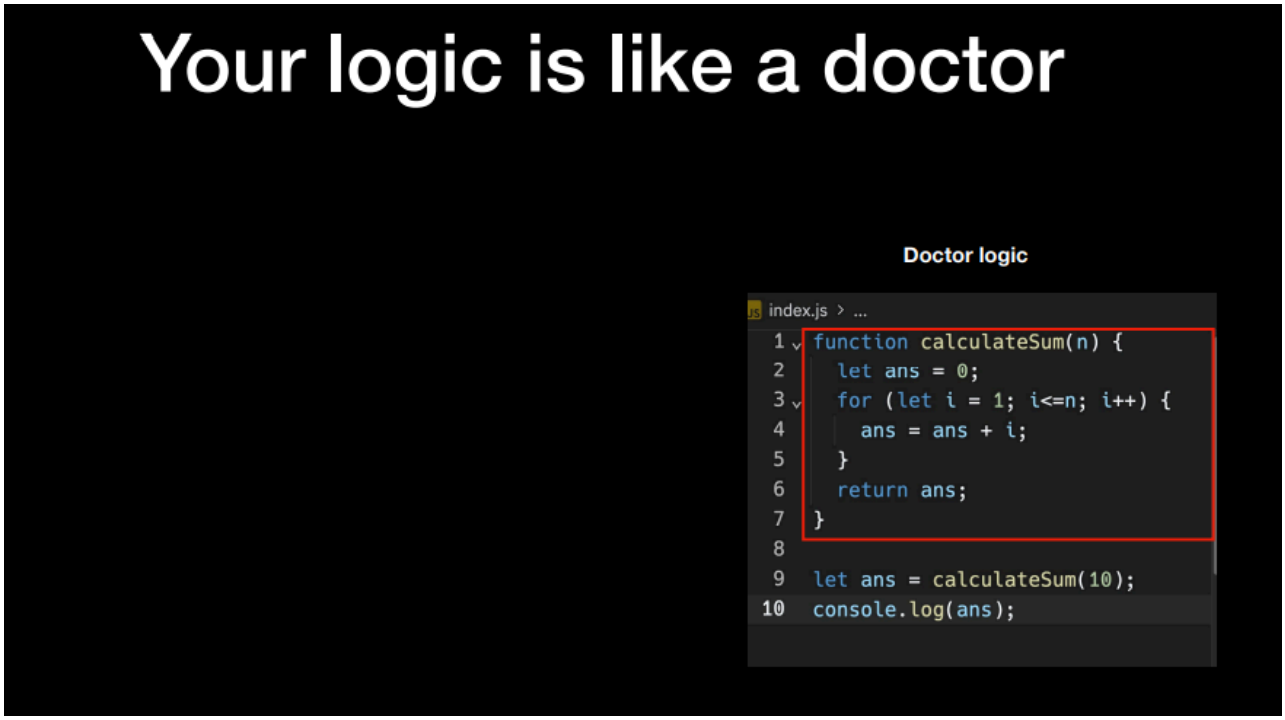


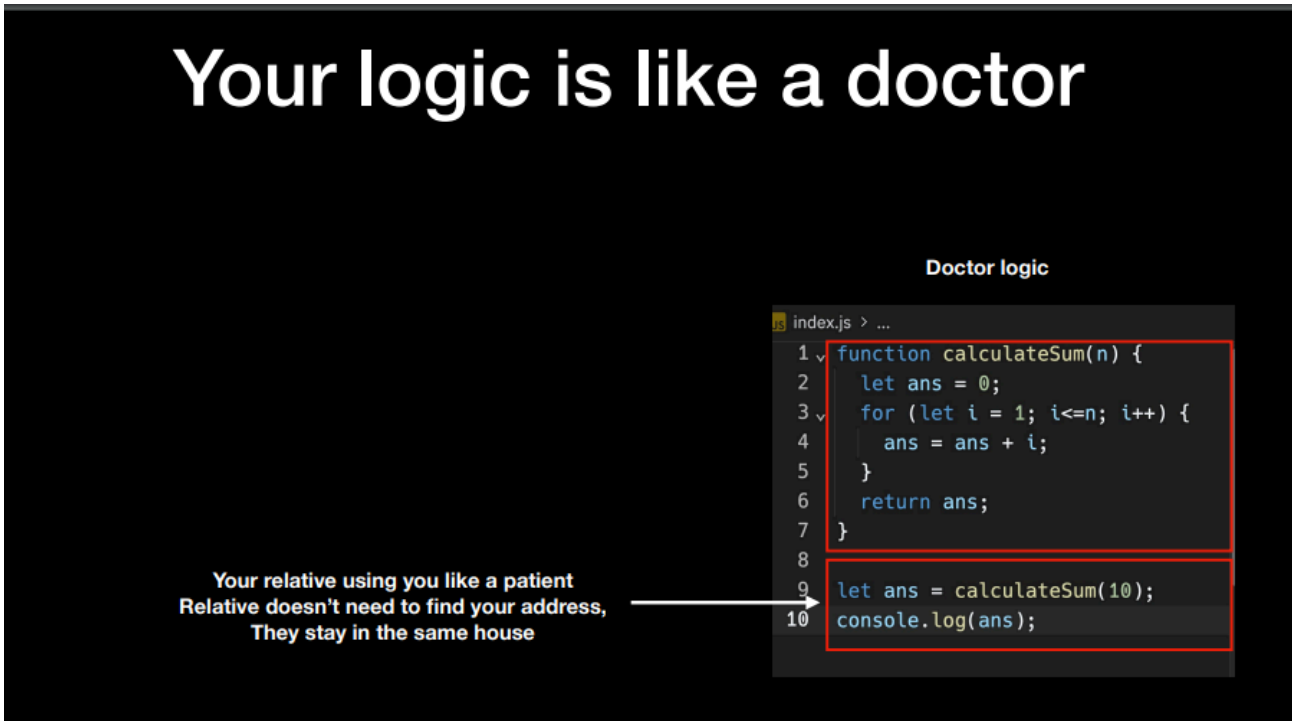
Go get medicine from chemist



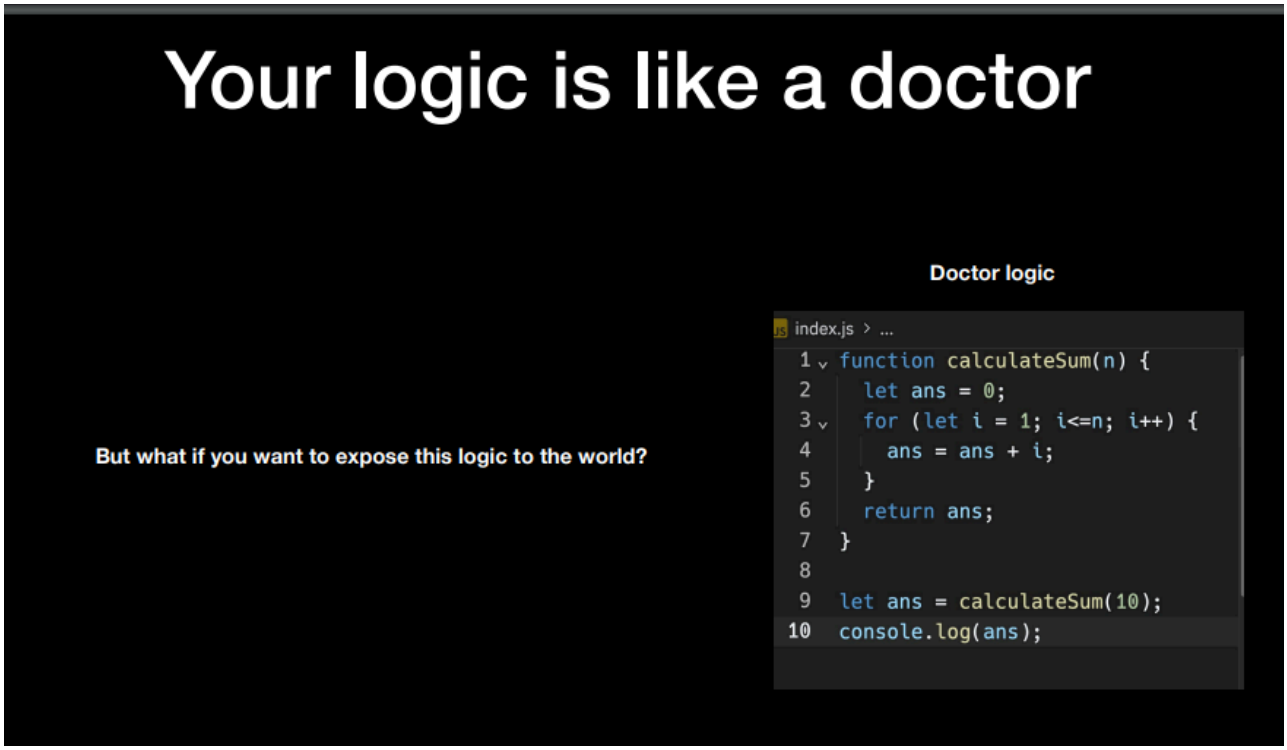
And then doctor will call when he/she is free.(doctor is single threaded) (they can attend only one person at a time)

Our logic is like a doctor

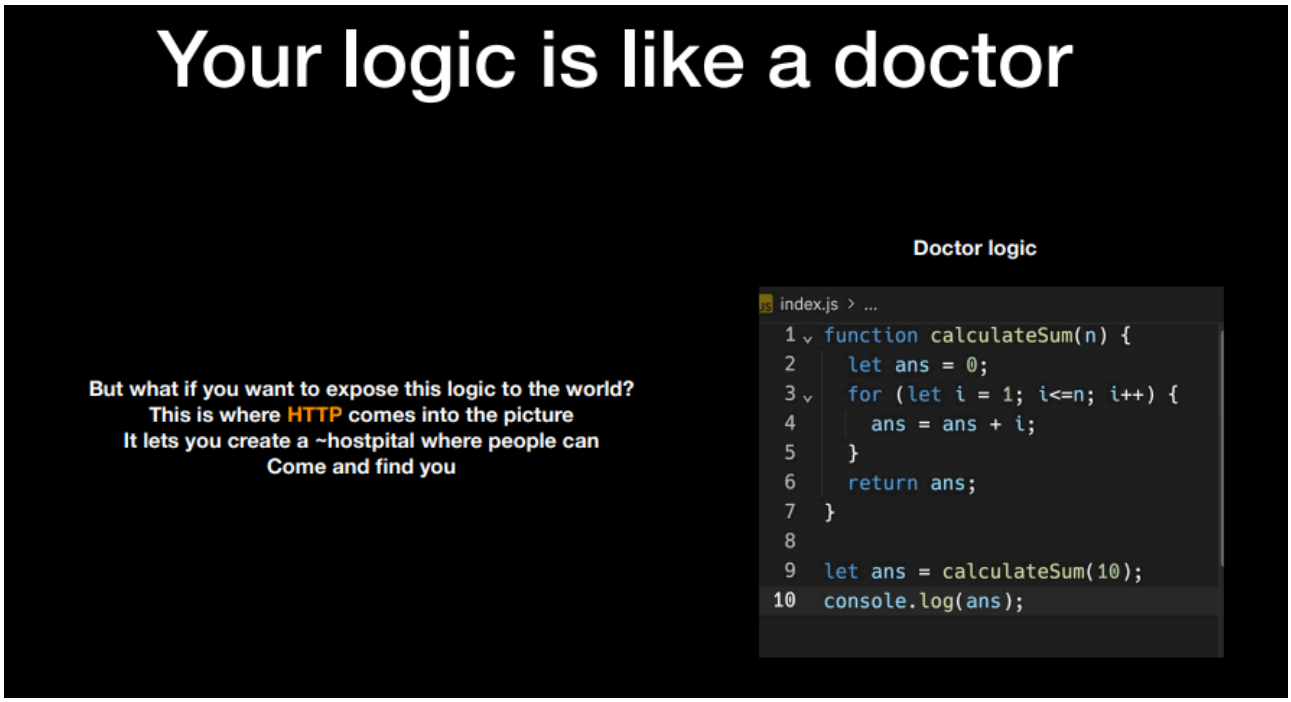




What if we want others to use our logic too.



We will use HTTP server.



Your logic is like a doctor

Question - How do I expose my doctor functionality
To other people?
How can they find me?

Ans - By creating an HTTP Server

Doctor logic

index.js > ...

```
1  function calculateSum(n) {
2    let ans = 0;
3    for (let i = 1; i<=n; i++) {
4      ans = ans + i;
5    }
6    return ans;
7  }
8
9  let ans = calculateSum(10);
10 console.log(ans);
```

Your logic is like a doctor

Question - How do I create an HTTP Server?

Ans - Express

Doctor logic

index.js > ...

```
1  function calculateSum(n) {
2    let ans = 0;
3    for (let i = 1; i<=n; i++) {
4      ans = ans + i;
5    }
6    return ans;
7  }
8
9  let ans = calculateSum(10);
10 console.log(ans);
```

Your logic is like a doctor

Question - How do I create an HTTP Server?

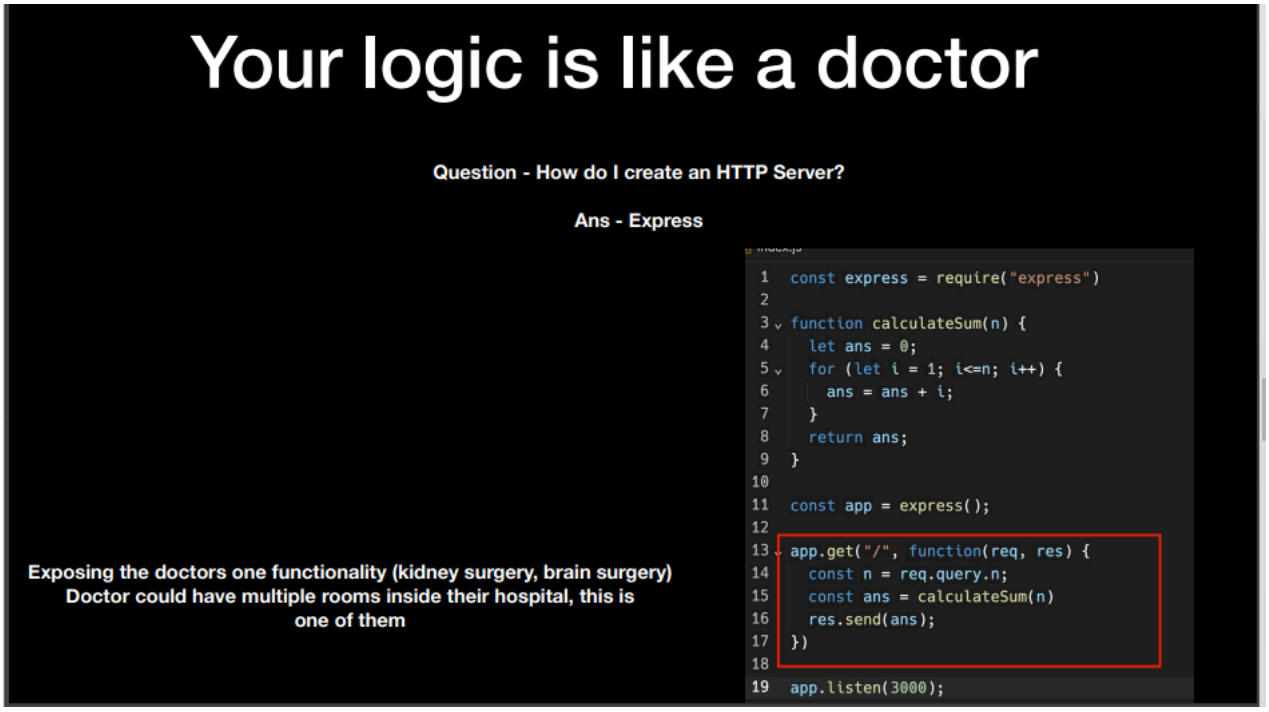
Ans - Express

index.js > ...

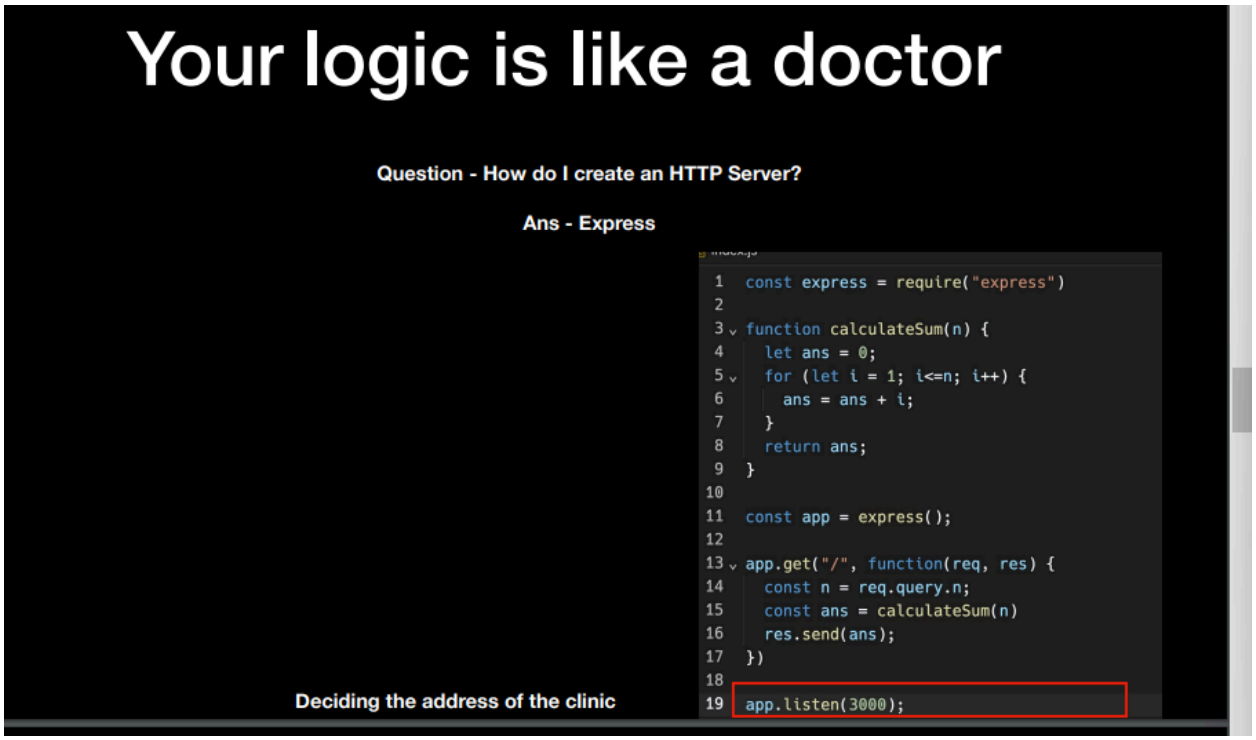
```
1  function calculateSum(n) {
2    let ans = 0;
3    for (let i = 1; i<=n; i++) {
4      ans = ans + i;
5    }
6    return ans;
7  }
8
9  let ans = calculateSum(10);
10 console.log(ans);
```

```
1  const express = require("express")
2
3  function calculateSum(n) {
4    let ans = 0;
5    for (let i = 1; i<=n; i++) {
6      ans = ans + i;
7    }
8    return ans;
9  }
10
11 const app = express();
12
13 app.get("/", function(req, res) {
14   const n = req.query.n;
15   const ans = calculateSum(n)
16   res.send(ans);
17 })
18
19 app.listen(3000);
```

If anyone comes on this route this logic should run (visit our hospital)



Deciding the address of the clinic. Any surgery query will come here in this address

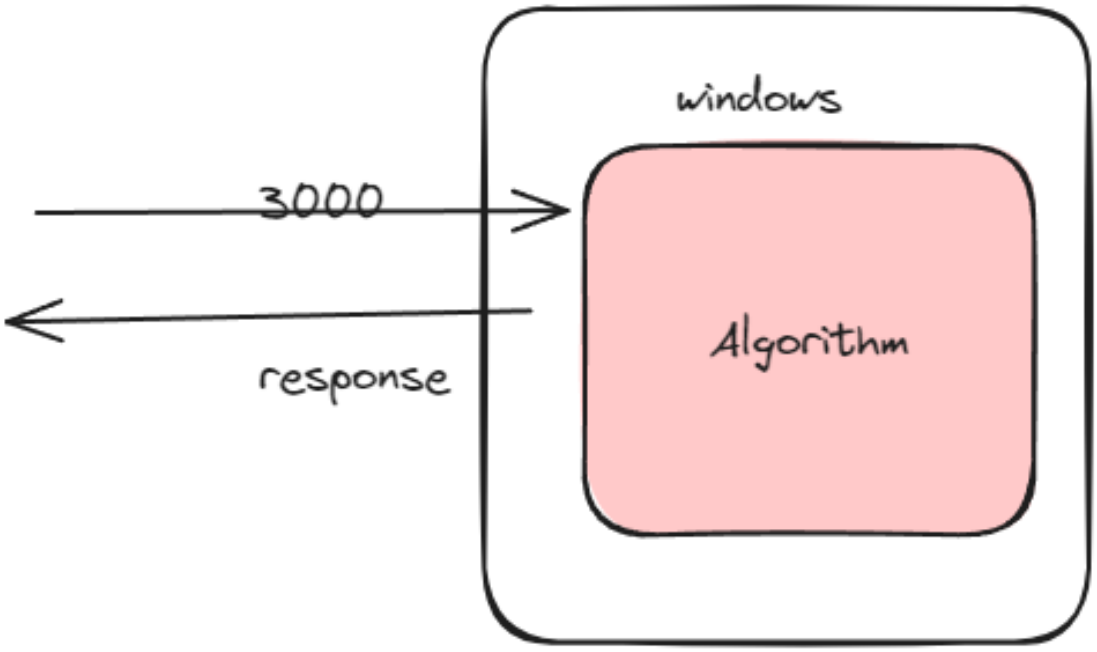


We can also have two doctors (two http server) in a hospital

Http can be create in any language we need to remember that the port must be unique.

Use Excalidraw to draw or explain an algorithm

How someone can excess the Algorithm from our system.

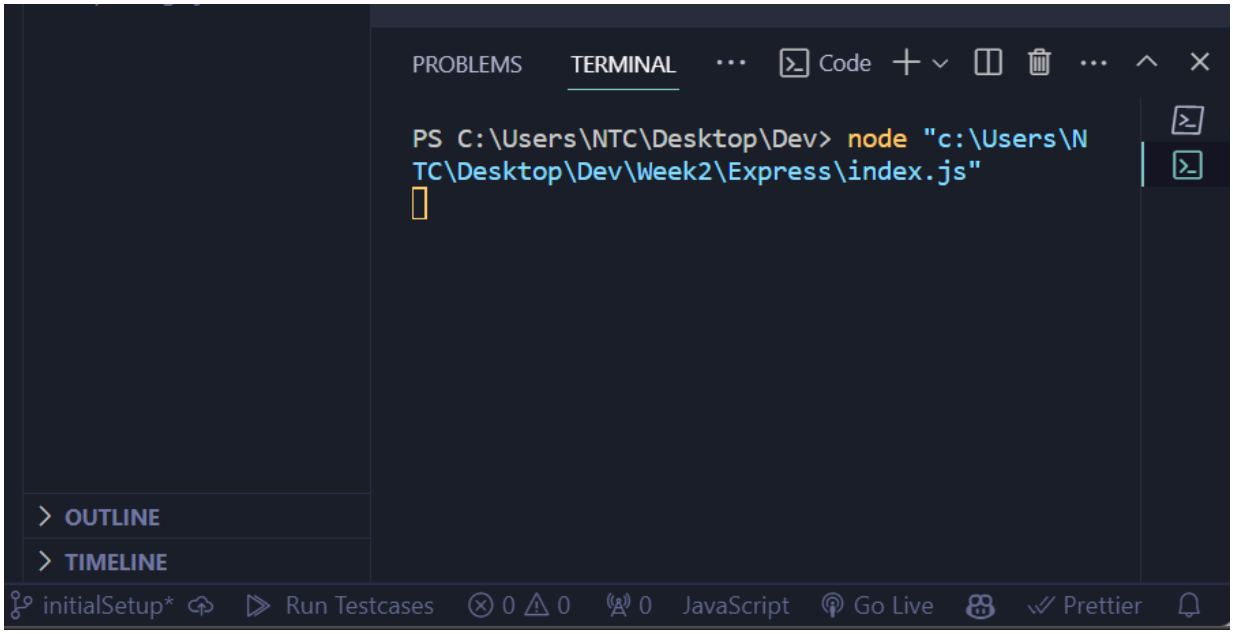


How do patient reach it??

Code

```
// creating an http server using an express
// express is not node default library like fs

const express = require('express');
// we have to bring it locallyy
// npm install express

const app = express();
// like creating a clinic

app.listen(3000);
//This is like listening to clinic( doctor getting a room )
```

We can see that our terminal is hung up

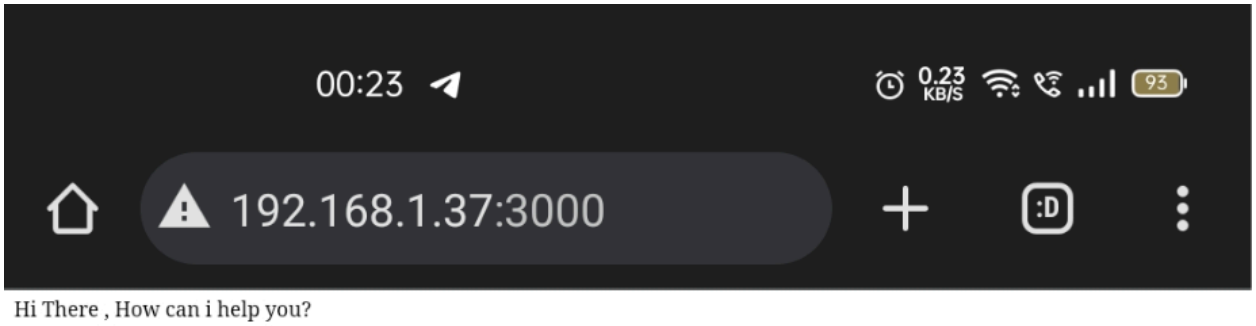


When we go to http://localhost:3000

We can say that the doctor has set up the clinic but currently, he serves(offers any service) nothing. This is the default express message

Let's see a basic http server

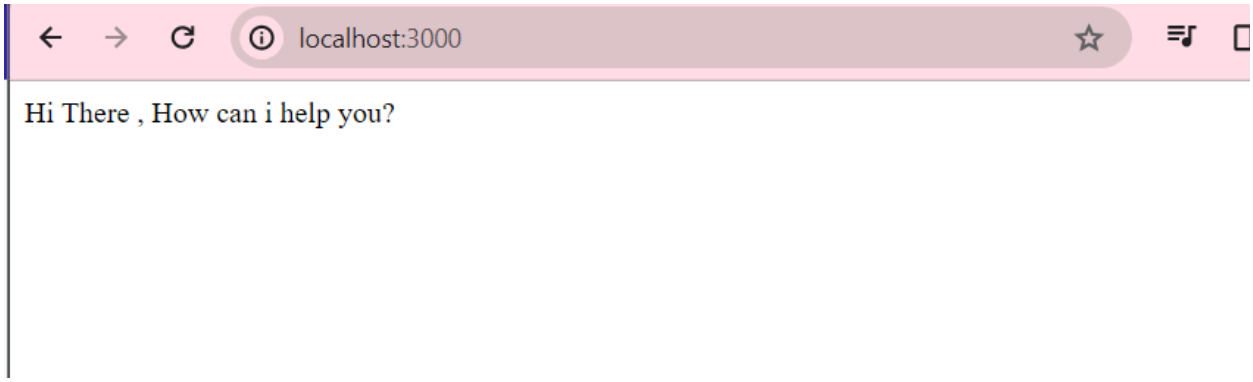
(We now allow people all over the world to access our logic although it does mostly nothing)(we haven't yet deployed it in the internet, but we can access on a local device (like phone))(by putting ip address)



By doing such we can delegate the task

```
const express = require('express');
const app = express();

// syntax of express: what route should I listen to and what should I do
when I get a request
// patient will be sent here until the doctor is free
//If three people come one by one doctor will attend to them
app.get('/', (req, res) => {
    res.send('Hi There , How can i help you?');
});

app.listen(3000);
```

Output in browser:



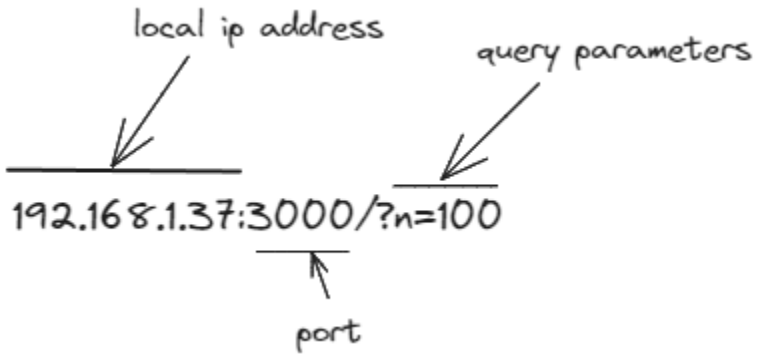
Passing Query parameter

```
const express = require('express');
const app = express();

function sum(n){
    // sum function require n , this is similar as doctor require x ray of
the patient to determine whether he/she has a broken bone or not
    let ans = 0;
    for(let i = 1; i <= n; i++){
        ans += i;
    }
    return ans;
}

app.get('/', (req, res) => {
    const n = req.query.n;//input we are sending to the server
    const sumOfN = sum(n);//calling sum function
    res.send('Hi There , your answer is '+ sumOfN);
});

app.listen(3000);
```

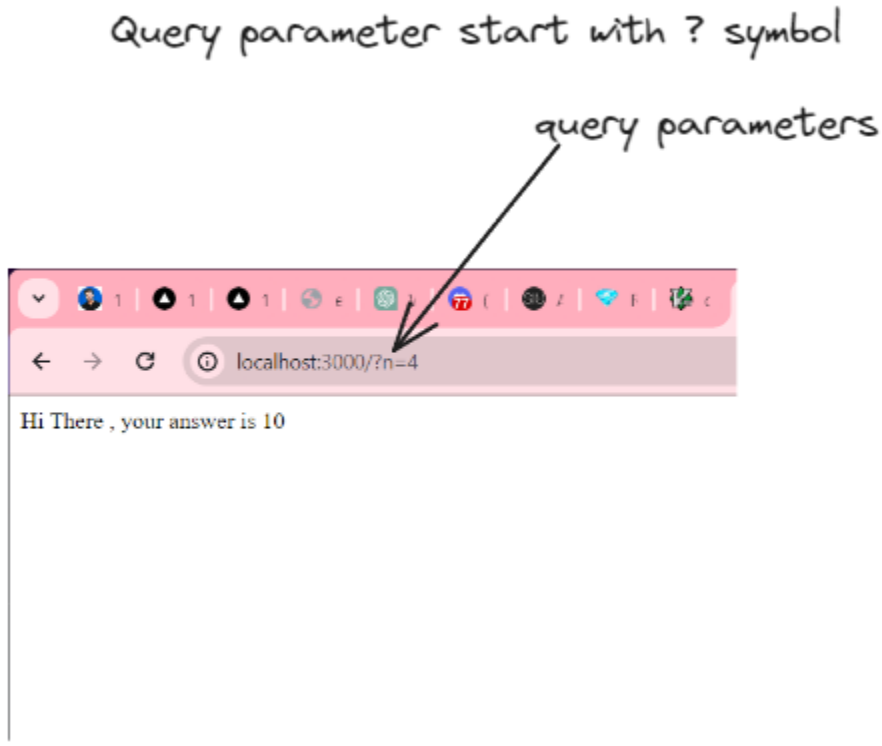
Output:

We can run this same on the phone by typing

in our browser

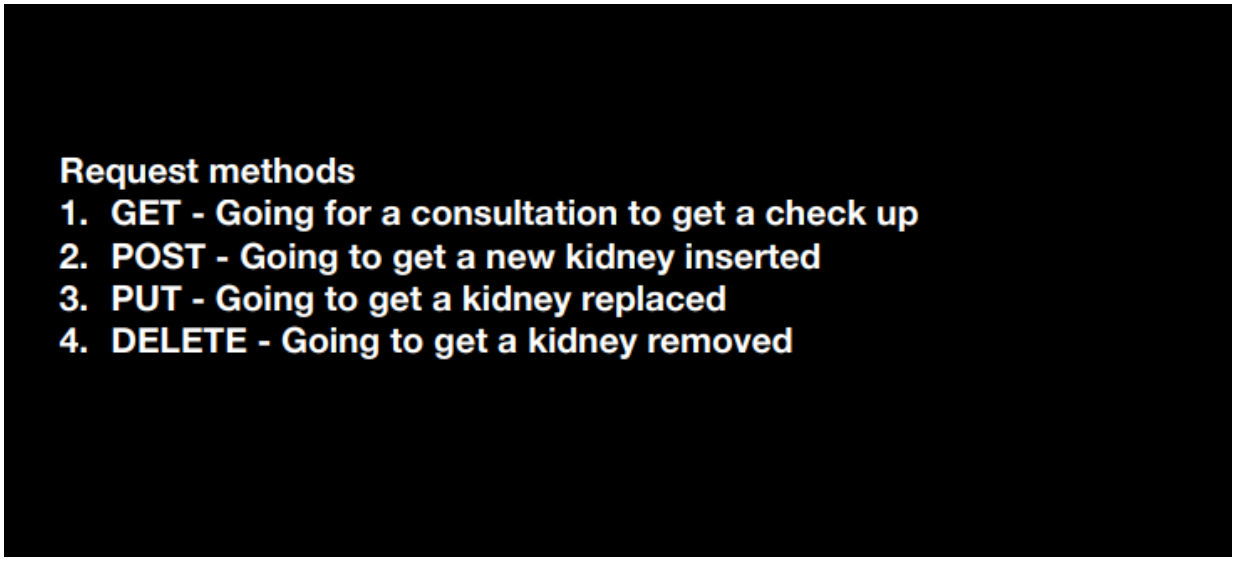


Output:



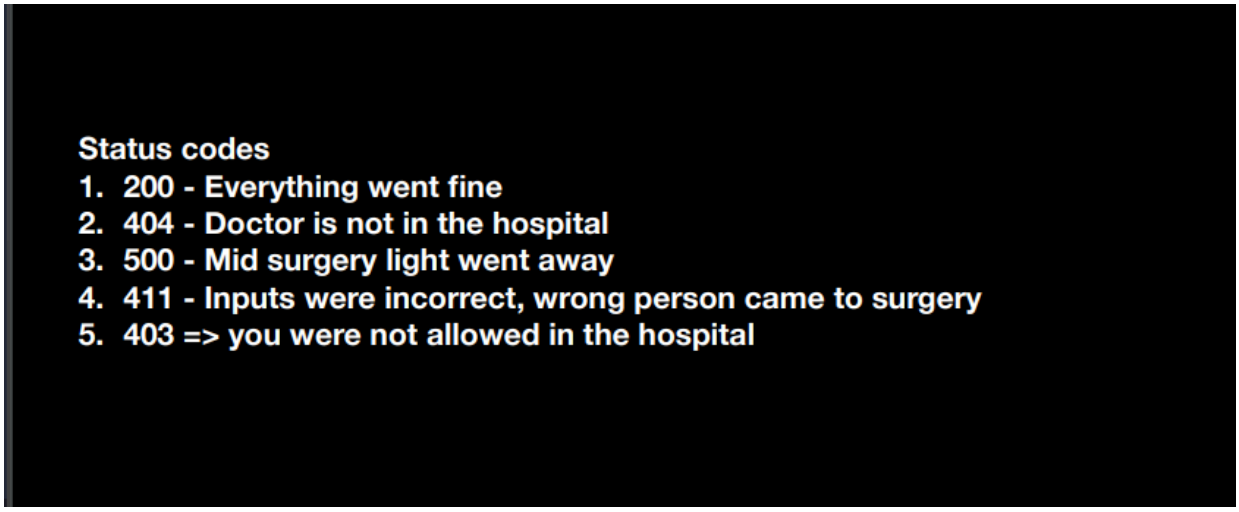
Request Methods

We are using the same Doctor example to understand the request methods



Status code

Default status code is 200



Application developers can follow these practices, but they could not it their choice and they could follow a different logic

Let's create an in-memory(not using any database **) hospital**

We need to create 4 routes (4 things that the hospital can do)

1. GET - User can check how many kidneys they have and their health
2. POST - User can add a new kidney
3. PUT - User can replace a kidney, make it healthy
4. DELETE - User can remove a kidney

We will first create an array of objects storing the information of the users.

```
var users = [{
    name: 'John',
    kidneys: [{
        healthy: false
        //One kidney not healthy
    },{
        healthy: true
        //One kidney healthy
    }]
}]

console.log(users[0]) // { name: 'John', kidneys: [ { healthy: false }, { healthy: true } ] }
```

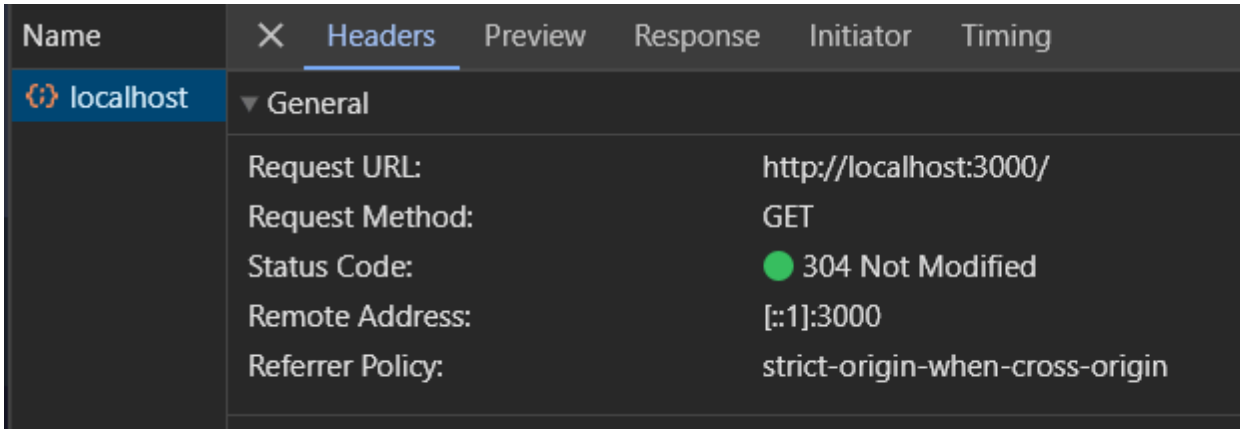
Let's create four routes

app.get()

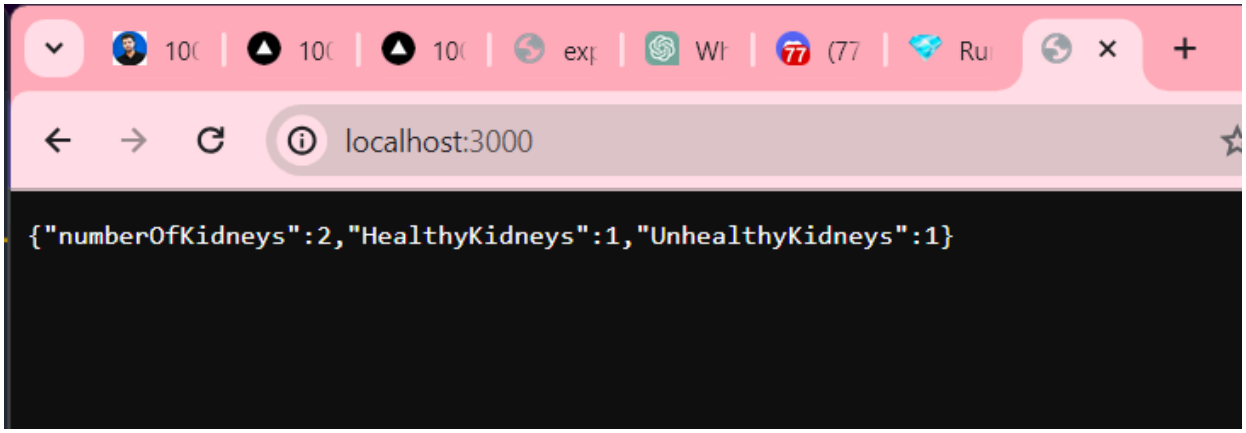
```
const express = require('express')
const app = express()

var users = [{
    name: 'John',
    kidneys: [{
        healthy: false
    },{
```

```
        healthy: true
    }]
}]

app.get('/', (req, res) => {
    const johnKidneys = users[0].kidneys;
    const numberOfKidneys = johnKidneys.length;
    // filter out the healthy kidneys
    let numberOfHealthyKidneys = 0;
    for (let i = 0; i < numberOfKidneys; i++) {
        if (johnKidneys[i].healthy) {
            numberOfHealthyKidneys++;
        }
    }
    const numberOfUnhealthyKidneys = numberOfKidneys -
numberOfHealthyKidneys;
    res.json({
        numberOfKidneys,
        HealthyKidneys: numberOfHealthyKidneys,
        UnhealthyKidneys: numberOfUnhealthyKidneys
    })
})
app.listen(3000)
```

We send **get** request by simply writing them on the browser



and it displays

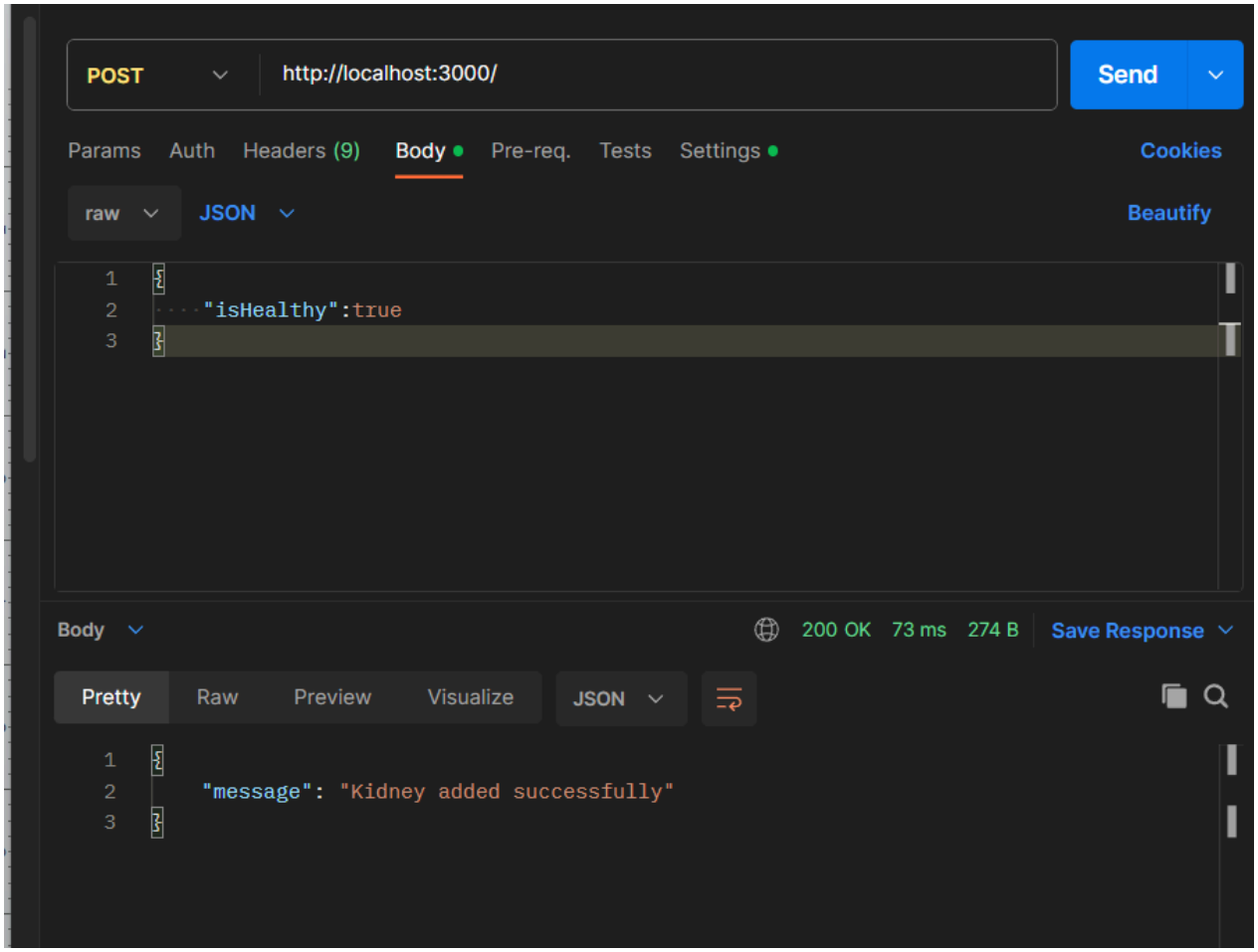


POST Endpoint

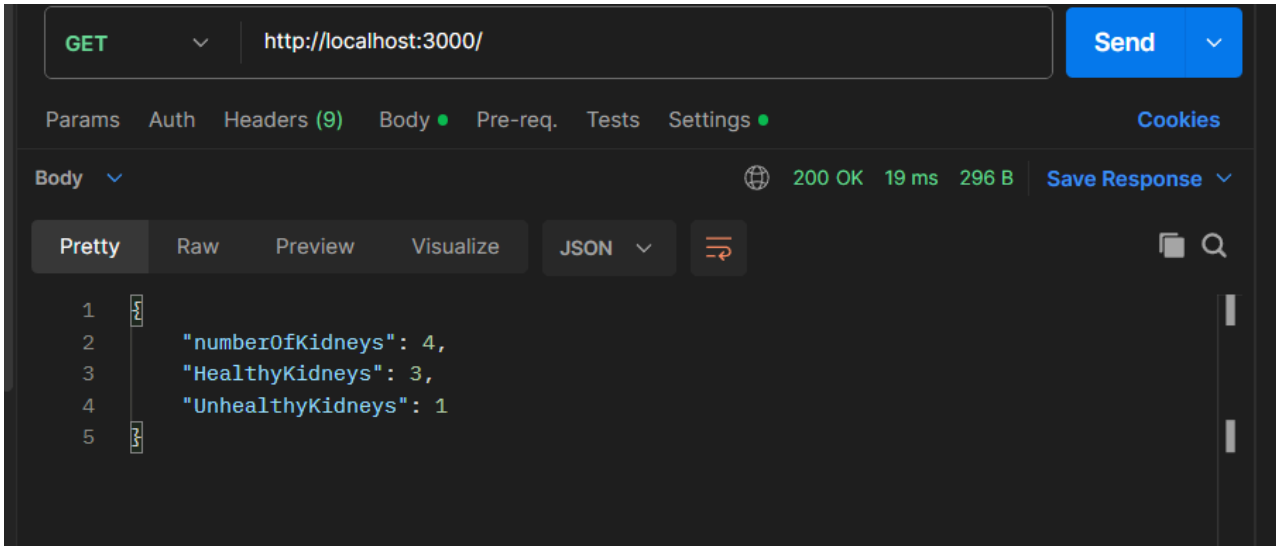
```
// we send data to body
// we can't use req.body directly we will know later
// middleware ( to parse and get json body we have to use
app.use(express.json())
app.use(express.json())
app.post("/", (req, res) => {
    const isHealthy = req.body.isHealthy;
    // they will send whether kidney healthy or not
    users[0].kidneys.push({
        // pushes a new kedney to the array
        healthy: isHealthy
    })
    res.json({
        message: "Kidney added successfully"
    })
    // they generally dont need respond when they send POST request (They
give input)
})
```

How do we send post request? We can also send it from the browser which we will see in later classes for now we will use Postman (sort of **simulators)**

We are sending **isHealthy** in the body because we make the post request catch the isHealthy



Now to confirm whether the new kidneys are added or not let do **get** a request

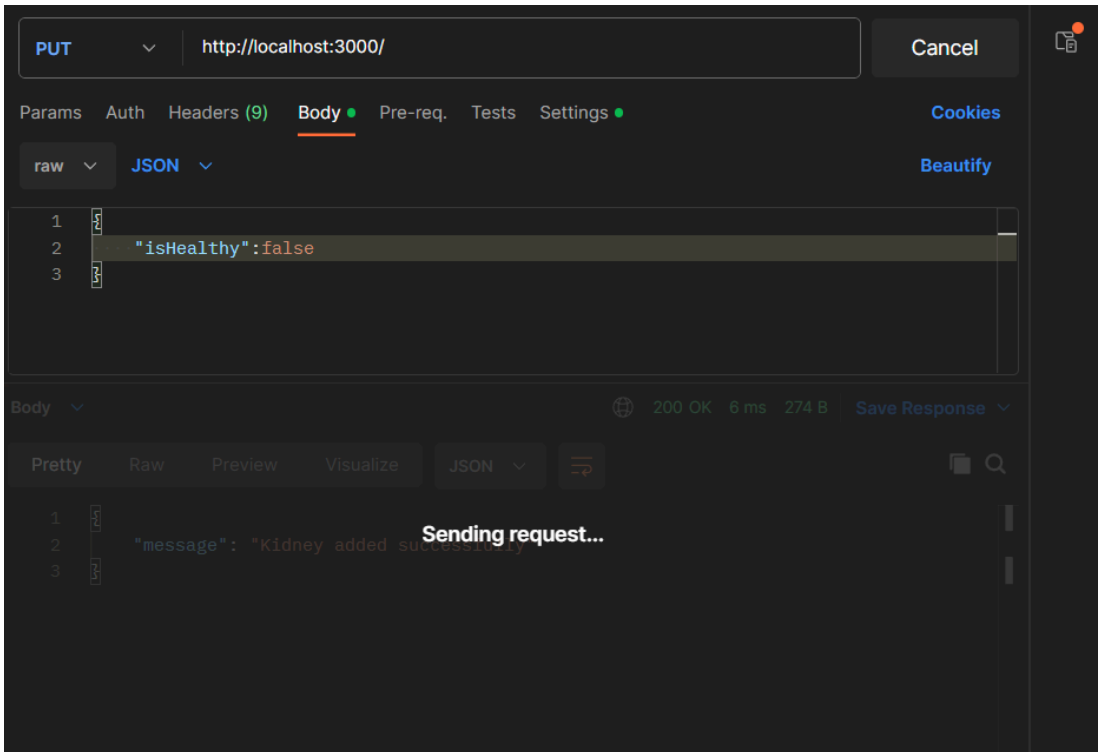


Any time we restart the server this no. will get to zero (that is why we need **Mongodb)**

It is the same as when we upload a LinkedIn post from mobile it is visible on the laptop as well.

Usually the common place where multiple clients can update their data is called **database.**

PUT

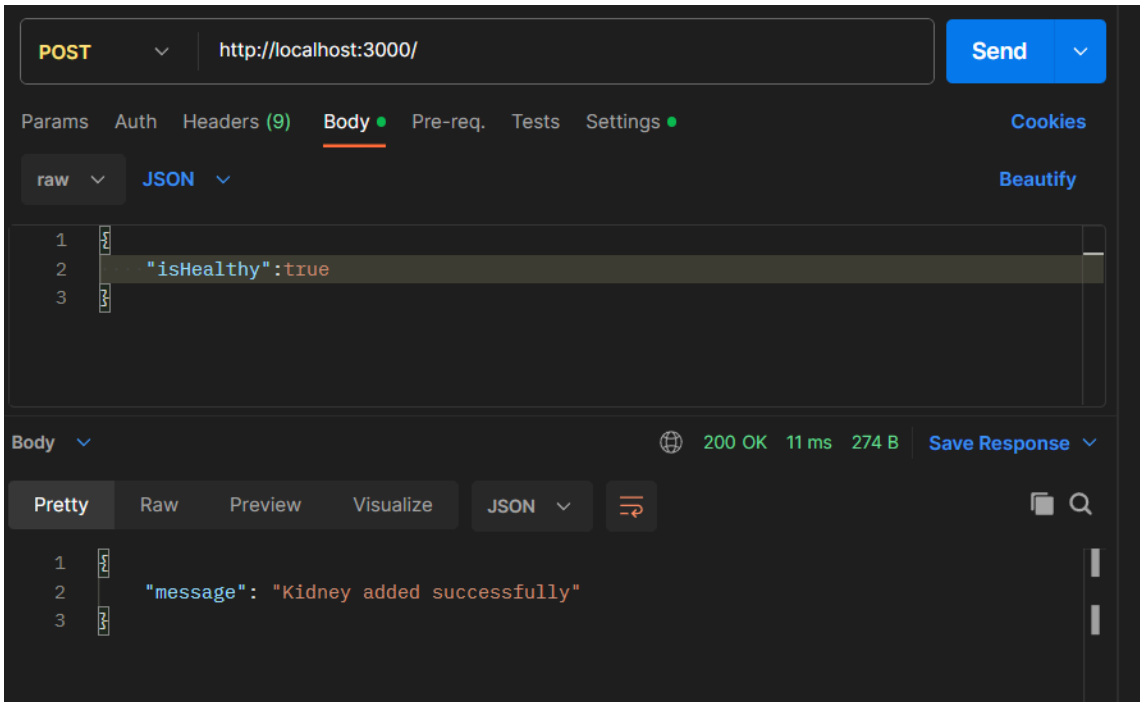


This shows sending request because we haven't send response.

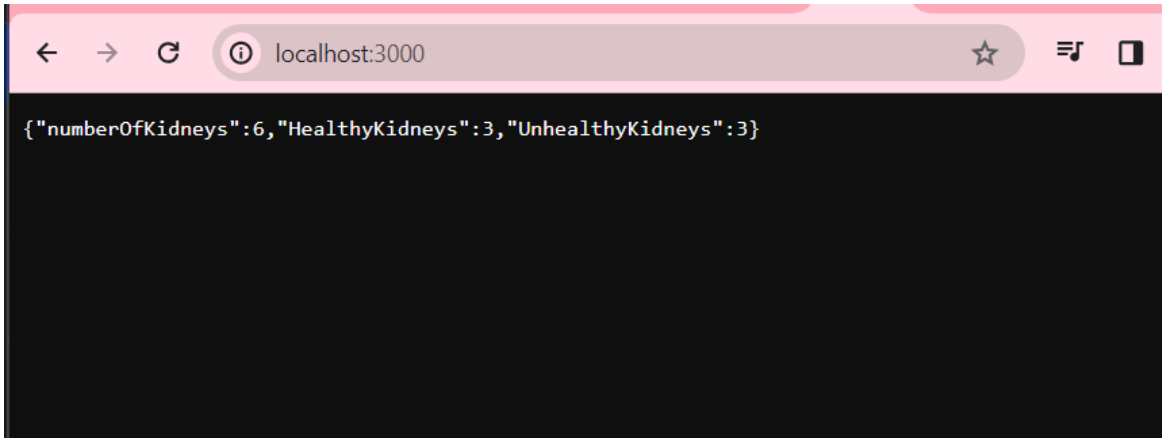
Code:

```
app.put("/", (req, res) => {
    // we need to update every kidney of user to healthy
    for( let i = 0; i < users[0].kidneys.length; i++) {
        users[0].kidneys[i].healthy = true;
    }
    res.json({
        message: "All kidneys are healthy"
    })
    // we can even send an empty data
    // if we don't send a response it will be pending forever (it will show sending requests)
})
```

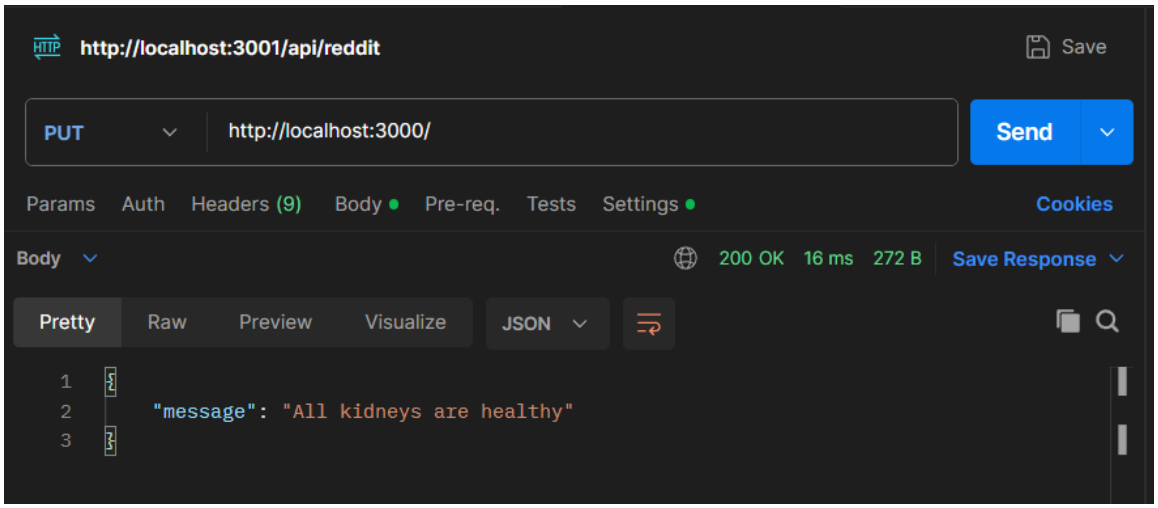
Sending **POST** request



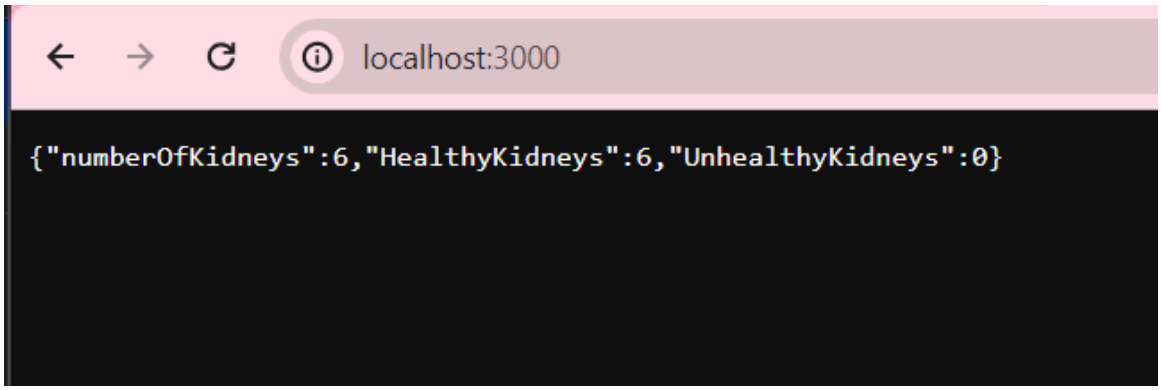
Then send the **GET** request



Now we will send the **PUT** request



Now again sending a **GET** request to verify if it took effect

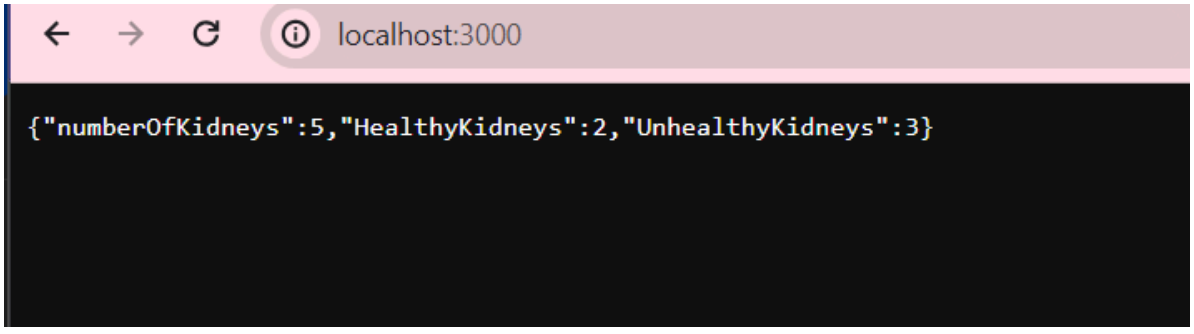


DELETE Endpoint

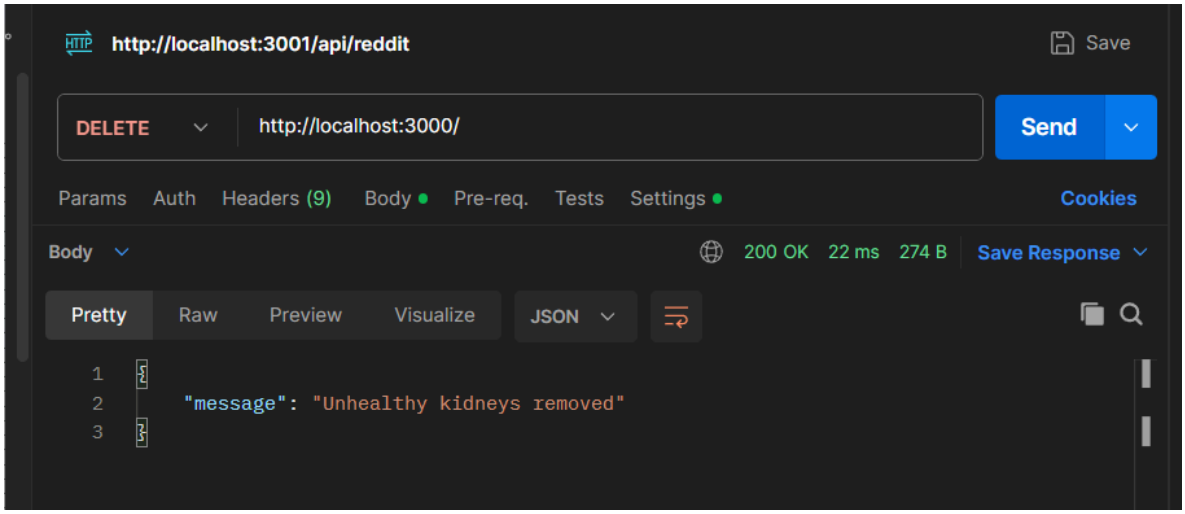
Code:

```
app.delete("/", (req, res) => {
    // we need to delete the unhealthy kidneys
    const newKidneys = [];
    for( let i = 0; i < users[0].kidneys.length; i++) {
        if (users[0].kidneys[i].healthy) {
            // pushing healthy kidney to newKidneys array
            newKidneys.push({
                healthy: true
            });
        }
    }
    users[0].kidneys = newKidneys;
    res.json({
        message: "Unhealthy kidneys removed"
    })
})
```

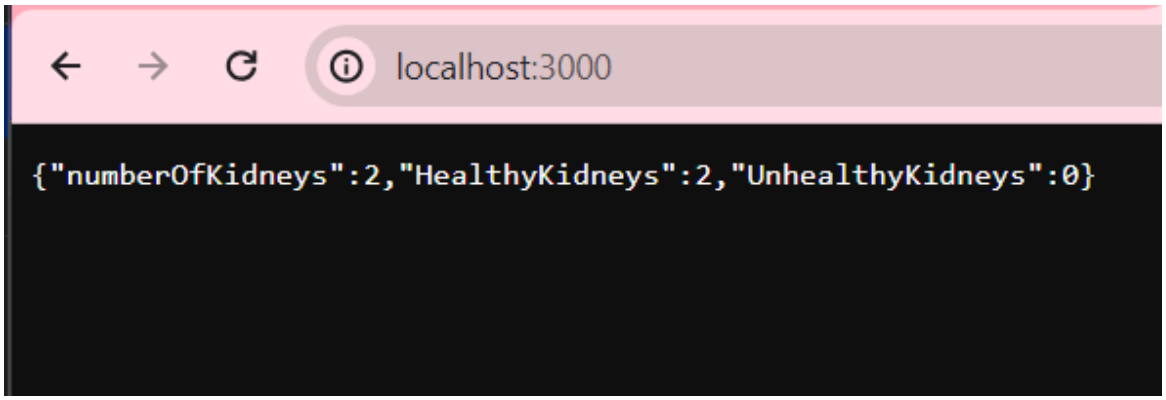
Sending a **GET request** after doing a bunch of **POST request .**



Now sending a **DELETE request** (this will delete all the unhealthy kidneys)



Now again sending a **GET request** to verify the result



Till now we have considered some ideal cases only.
Now if we send some gibberish data suppose during the **post** request then the server will crash and the user will send some errors.

What should happen if they try to delete when there are no kidneys?
What should happen if they try to make a kidney healthy when all are already healthy?

ON FRONTEND We check status codes, suppose its a 200 status code we show nothing but if there is 404 **Then we have to show that there was some error. Hence adding a status code is good practice.**

Code:

```
const express = require('express')
const app = express()
app.use(express.json())

// mini database
var users = [{
    name: 'John',
    kidneys: [{
        healthy: false
    }]
}]

// query parameters famous for get request
app.get('/', (req, res) => {
    const johnKidneys = users[0].kidneys;
    const numberOfKidneys = johnKidneys.length;
    // filter out the healthy kidneys
    let numberOfHealthyKidneys = 0;
    for (let i = 0; i < numberOfKidneys; i++) {
        if (johnKidneys[i].healthy) {
            numberOfHealthyKidneys++;
        }
    }
    const numberOfUnhealthyKidneys = numberOfKidneys -
numberOfHealthyKidneys;
    res.json({
        numberOfKidneys,
        HealthyKidneys: numberOfHealthyKidneys,
        UnhealthyKidneys: numberOfUnhealthyKidneys
    })
})

// we send data to body
```

```
// we can't use req.body directly we will know later
// middleware ( to parse and get json body we have to use
app.use(express.json())

app.post("/", (req, res) => {
    const isHealthy = req.body.isHealthy;
    // they will send whether kidney healthy or not
    users[0].kidneys.push({
        // pushes a new kedney to the array
        healthy: isHealthy
    })
    res.json({
        message: "Kidney added successfully"
    })
    // they generally dont need to respond when they send POST request
(They give input)
})

app.put("/", (req, res) => {
    // we need to update every kidney of user to healthy
    for( let i = 0; i < users[0].kidneys.length; i++) {
        users[0].kidneys[i].healthy = true;
    }
    res.json({
        message: "All kidneys are healthy"
    })
    //We can even send empty data
    //If we don't send a response it will be pending forever (it will show
sending requests)
})

app.delete("/", (req, res) => {
    //We need to delete the unhealthy kidneys

    //We should send 411 status code(wrong input) (if they have a kidney
healthy then there is no point in going to surgery/delete the kidney)
```

```
    if(isThereAtLeastOneUnhealthyKidney()){
        const newKidneys = [];
        for( let i = 0; i < users[0].kidneys.length; i++) {
            if (users[0].kidneys[i].healthy) {
                // pushing healthy kidney to newKidneys array
                newKidneys.push({
                    healthy: true
                });
            }
        }
        users[0].kidneys = newKidneys;
        res.json({
            message: "Unhealthy kidneys removed"
        })
    }else{
          res.status(411).json({
              message: "No unhealthy kidneys"
          })
    }
})

function isThereAtLeastOneUnhealthyKidney(){
    let atLeastOneKidneyHealthy = false;
    for( let i = 0; i < users[0].kidneys.length; i++) {
        if (!users[0].kidneys[i].healthy) {
            //If at least one kidney is unhealthy
            atLeastOneKidneyHealthy = true;
        }
    }
    return atLeastOneKidneyHealthy;
}

app.listen(3000)
```