My First Express:: Continued

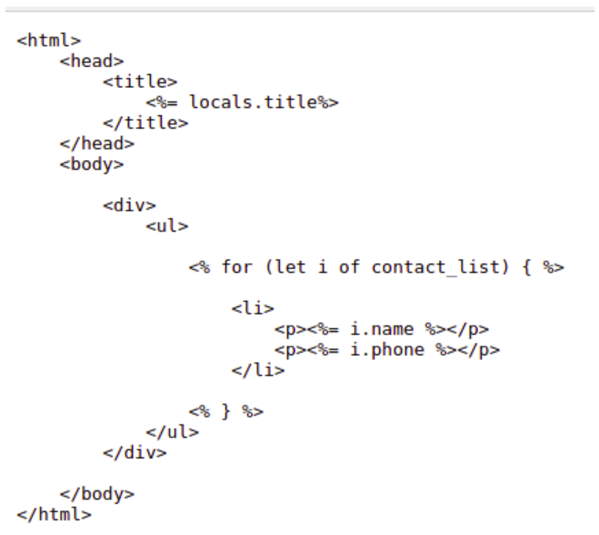
Introduction

We will be creating a form on the page, we will be entering data into that form and saving that data onto the server but not permanently. We will be able to delete a contact from the list of contact forms.

Creating a Contact List

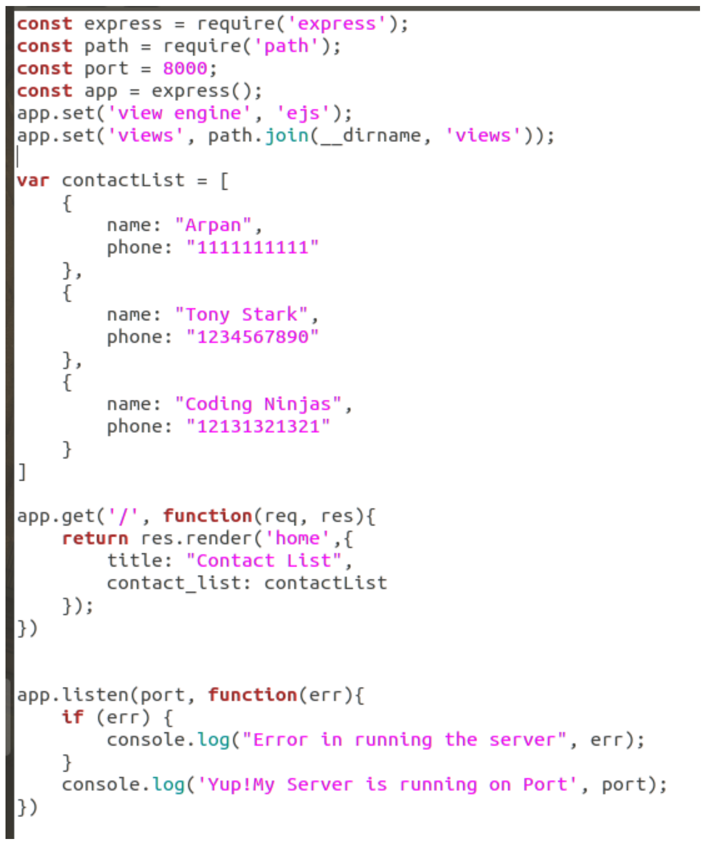
- We will be creating an array of contacts that will contain the name and phone number of the user in **{ index.js }** file.
- We need to send the phone number on the home page. For that, we just need to pass the contact list as one of the contexts in the form of key and value.
- For each item in the contact list, we need a list item in the { **home. ejs** } file for name and phone number.

{ **home. ejs** }



```
<html>
    <head>
        <title>
            <%= locals.title%>
        </title>
    </head>
    <body>

        <div>
            <ul>

                <% for (let i of contact_list) { %>

                    <li>
                        <p><%= i.name %></p>
                        <p><%= i.phone %></p>
                    </li>

                <% } %>
            </ul>
        </div>

    </body>
</html>
```

{ index.js }



```
const express = require('express');
const path = require('path');
const port = 8000;
const app = express();
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

var contactList = [
    {
        name: "Arpan",
        phone: "1111111111"
    },
    {
        name: "Tony Stark",
        phone: "1234567890"
    },
    {
        name: "Coding Ninjas",
        phone: "12131321321"
    }
]

app.get('/', function(req, res){
    return res.render('home',{
        title: "Contact List",
        contact_list: contactList
    });
})


app.listen(port, function(err){
    if (err) {
        console.log("Error in running the server", err);
    }
    console.log('Yup!My Server is running on Port', port);
})
```

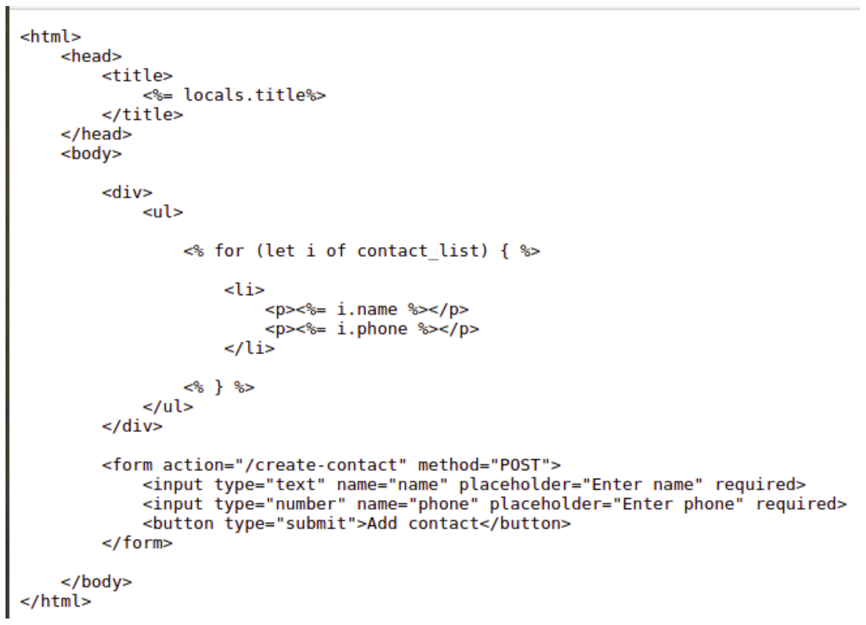
<u>NOTE</u>:

- *Camel case is the naming convention that is followed by Javascript.*

Sending Data to The Server

- We have created the contact list and rendered it on the web page, the next step is to add and delete something from the list.
- We have to create a form inside the **{ home. ejs }** value from wherein we can take the data from the user in the browser which gets appended to the existing contact list.
- The name attribute is used for naming the keys of the data that needs to be sent.

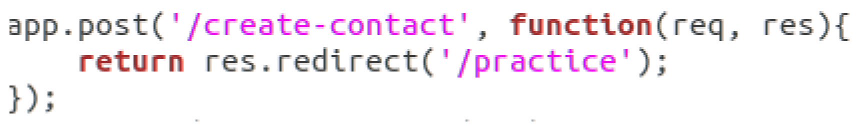
{ home. ejs }



```
<html>
    <head>
        <title>
            <%= locals.title%>
        </title>
    </head>
    <body>

        <div>
            <ul>

                <% for (let i of contact_list) { %>

                    <li>
                        <p><%= i.name %></p>
                        <p><%= i.phone %></p>
                    </li>

                <% } %>
            </ul>
        </div>

        <form action="/create-contact" method="POST">
            <input type="text" name="name" placeholder="Enter name" required>
            <input type="number" name="phone" placeholder="Enter phone" required>
            <button type="submit">Add contact</button>
        </form>

    </body>
</html>
```

- Whenever we submit a form inside the file **{ home. ejs }** that form usually consists of the user contact details.
- After submitting each form we are submitting specific contact details that we want to append in the list that is viewed on the home page.
- We have to create a controller for the form that is being submitted on the profile page inside the **{ index.js }** file.

{ index.js }



```
app.post('/create-contact', function(req, res){
    return res.redirect('/practice');
});
```

- Redirect is a function that takes the user to the specific route that is mentioned inside the redirect function.
- We have to create a route { **practice. ejs** } where we are redirecting the user after submitting the form.

{ **practice. ejs** }

```
<html>
    <head>
        <title>
            <%= title %>
        </title>
    </head>
    <body>
        <h1>Playground</h1>

        <ul>
            <% for (let i=1; i<10; i++) { %>
            <li>
                <%= i %>
            </li>
            <%}%>
        </ul>

        <% if (true){ %>
            <h1>It's true!</h1>

        <% }else{ %>
            <p>It's false :(</p>
        <% } %>


    </body>
</html>
```

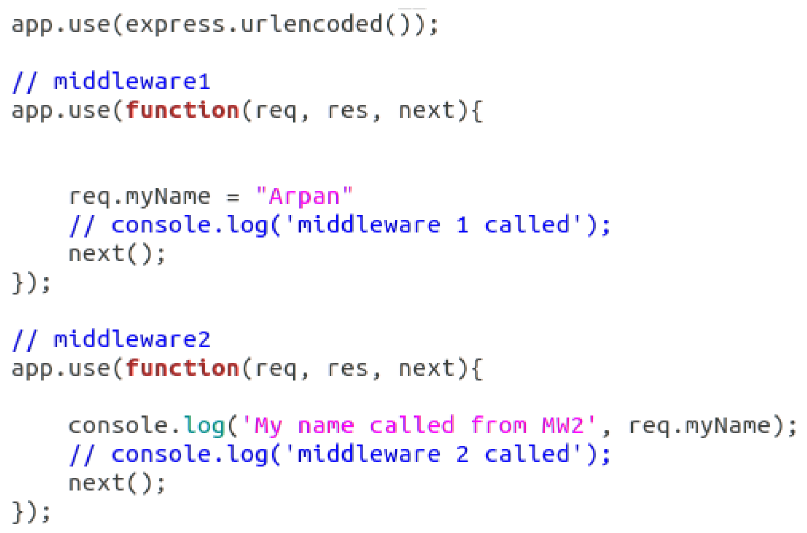
Parsing Form Data:: Introduction

- The name attribute is the most important one when we submit the form.
- When the data is sent from the form to the server, it is encoded in the form of a string. To analyze it and decode it into the form of an object **{ key-value pair }** we need a parser.
- The parser takes the data from the browser.
- The parser creates a key body inside the request { that is an object } and the value of that key will be the data coming from the browser.
- The parser is already present in the express, we just have to use it.

Middleware: What and How

- Middleware is a function that has access to the request and response that can analyze or preprocess the data present inside the request and response and then parse it on or throw an error if there is any.
- We need middleware because sometimes there are some common things we need to do, such as preprocessing the data.
- **App. use** is signifying the middleware and **express.urlencoded** takes the request and reads or analyzes the data.
- We will create our own middleware in the **{ index.js }** file.
- The next argument passes on whatever changes have been done and calls the next middleware if there is any, if not it will pass it on to the controllers.

{ index.js }



```
app.use(express.urlencoded());

// middleware1
app.use(function(req, res, next){


    req.myName = "Arpan"
    // console.log('middleware 1 called');
    next();
});

// middleware2
app.use(function(req, res, next){

    console.log('My name called from MW2', req.myName);
    // console.log('middleware 2 called');
    next();
});
```

- Whenever the request comes in middleware1 and middleware2 is called. When it goes to the next function it takes the user to the controllers.
- Middleware can also be used to manipulate the request or response data.

Accessing the Static Files

- Static files provide functionality using Javascript, they beautify the page using images and CSS.
- We need to include static files using middleware.
- For static files, we need to create a folder of **assets** that will internally contain multiple folders for **CSS, images, Javascript,** and **fonts**.
- We will create a CSS file and Javascript file inside the CSS folder and JS folder.
- We have to also include those CSS and JS files inside the { **home.ejs** } file.

A Quick Revision

- We created the file **{ index.js }.**
- We did **npm init** to set up the project.
- We need to install the express server using the command { npm install express } in the terminal.
- We have started the express server by requiring the library inside the **{ index.js }** file.
- We have chosen a port and we have set the firing of the express app using express().
- We told the app to listen on the given port.
- We handled the error while running the app.
- We have sent the data from the server to the ejs file using views { EJS }.
- We have chosen EJS because it is similar to writing JS and the HTML remains the same.
- We have installed ejs using the command { npm install ejs } in the terminal.

- We told express to use EJS as the view engine.
- We have to look out to the views folder for views.
- We have to specify the path for the views folder.
- We have set up the EJS file that will be rendering the HTML.
- We have passed on the data in the context.
- We have set up the middleware to read a request from the form.
- We have included the middleware that will read the form data then parses it into keys and values.
- When we receive those keys and values it is into request. body that will be saved into the contact lists.
- We have included the static files using the middleware.

__NOTE__:

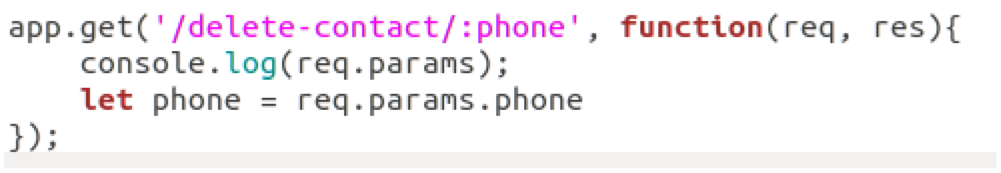
- ***As a mini assignment design the contact list***

Query and String Parameters: Beginning

- We have to create a delete button for deleting the contact.
- We have to create a controller with the route and a link that points towards the delete button or a form that sends the data to it.
- We need to find the item that matches the phone number. If it matches the phone number, we need to delete it.
- For deleting in the array we will use a function called splice.
- To get the value of the phone there are two ways - Query params and String params.
- For String params, we need to tell beforehand to the controller that you will be receiving some number that will be the variable part. Eg - { /delete-contact/**10** (String Params)}

- For Query params, we do not know beforehand what all query params we are receiving. Eg - { delete-contact/**?id = 10** (Query Params)}.
- We can chain query params as if we want to for eg - { /delete-contact/?id=10&name="anjali" }

{ index.js }

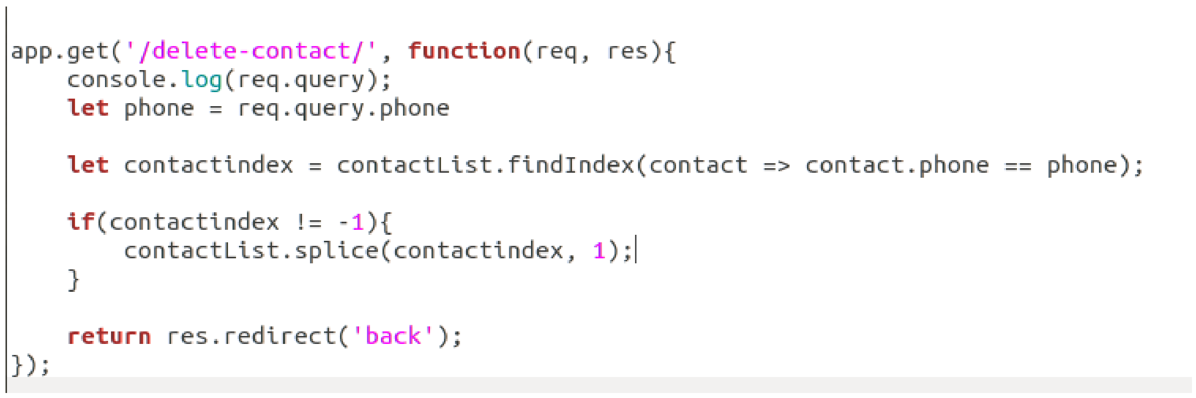


```
app.get('/delete-contact/:phone', function(req, res){
    console.log(req.params);
    let phone = req.params.phone
});
```

Deleting A Contact

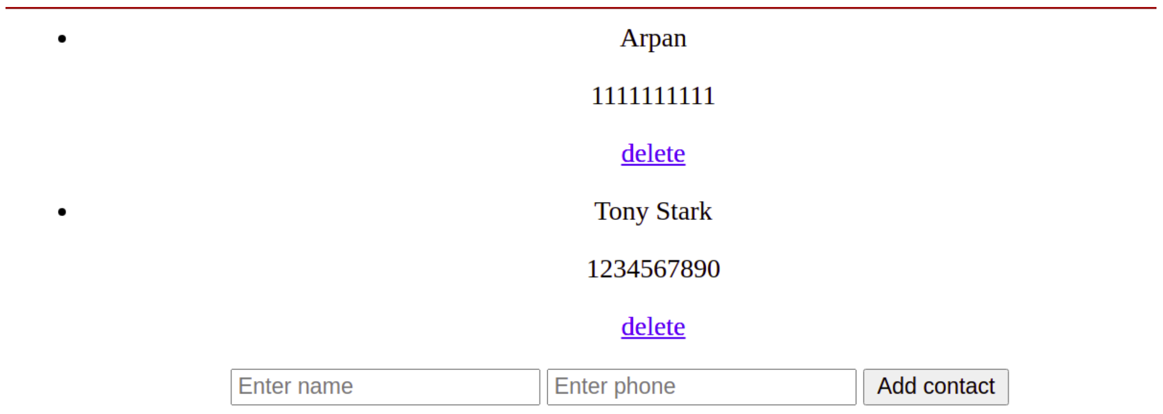
- We need to iterate over the contact list array and find the index of the matching phone number if it exists and then remove it from the array.
- The function to iterate over the list of contacts and find the index is **{ findIndex }.**

{ index.js }



```
app.get('/delete-contact/', function(req, res){
    console.log(req.query);
    let phone = req.query.phone

    let contactindex = contactList.findIndex(contact => contact.phone == phone);

    if(contactindex != -1){
        contactList.splice(contactindex, 1);
    }

    return res.redirect('back');
});
```

***<u>EXTRA</u>* - You check the page for your reference.**



- Arpan

  1111111111

  delete
- Tony Stark

  1234567890

  delete

Summarizing

- We have created the form and sent the data to the server.
- We have beautified the app by using static files
- We have stored the data from the form in the variable - fetch from that data, show it on the web page, delete from the variable and update the page accordingly.
- We have also studied params and middleware.