POST – collecting form data

Welcome to this lab activity

In this lab activity, you will explore how to collect data from your existing form using the POST request method. This contrasts with the use of the GET request method that you looked at in the previous lab. You will illustrate this using a new form that collects user registration details.

Task 1: Make a copy of your previous lab

You will start with your previous lab code.

1. Make a copy of the folder `08_get_form_data` from your previous lab and call it `09_post_form_data`.
2. Open the new folder in Visual Studio Code.
3. Run the code and confirm that you can see the homepage and about page (refer to the previous lab if you need a reminder of how to do this).

Task 2: Add the register form

Let's now create the register form. This will capture the first name and last name of the user in an HTML form and allow the user to submit the form to the web server. In this case, you will use the `POST` HTTP method. If you remember, the GET method passed the contents of the form to the web server in the URL, for example:

```
/search_result?search_text=carrots
```

In contrast, the `POST` method passes the contents of the form to the web server in the body of the HTTP request. This means the form contents are not visible in the browser. This is ideal where you are submitting confidential information, such as user details or passwords. So, it is a suitable approach for a user registration form, such as the one you are about to create.

4. Add a new file `register.ejs` to the views folder.
5. Add the following contents to `register.ejs`:

```
<!doctype html>
<html>
  <head>
    <title>This is the register page title!</title>
  </head>
  <body>
    <h1> This is the register page </h1>
    <form method="POST" action="/registered">
        First Name:<input id="first" type="text" name="first"/>
        Last Name:<input id="last" type="text" name="last" />
        <input type="submit" value="OK" />
    </form>
</body>
</html>
```

Notice that the method on the form is `POST` and the action is `/registered`. This means that when the form is submitted, the web server will get a `POST` request to the `/registered` route. So, you will need to handle this `POST` request.

Task 3: Handle the POST request

So, let's handle that POST request.

6. In main.js, add the following route handlers:

```
router.get("/register", (req,res) => {
    res.render("register.ejs");
});

router.post("/registered", (req,res) => {
  res.send(req.body)
});
```

The first one handles the `/register` route and shows the register form. This uses the Express `get()` method.

The second one handles the `/registered` route and extracts the contents of the form from the body of the request. This uses the Express `post()` method().

Task 4: Setting up the body parser

Express needs additional direction to extract the contents of a POST method. You will add a new middleware function to parse the body contents before you handle the routes.

7. In index.js, add the `urlencoded` middleware function before your route handler code:

```
// Set up the body parser
app.use(express.urlencoded({ extended: true }));

// Load the route handlers
const mainRoutes = require("./routes/main");
app.use('/', mainRoutes);
```

It is important that you perform the urlencoded() call before you perform the route handling so that the body is parsed, ready for the route handlers to work with.

Looking deeper

The above code is a bit mysterious, so you may want to dig further. The `express.urlencoded()` function looks at the contents of the body of the HTTP request and parses it, converting it to a JavaScript object. In this case, we are telling Express that the contents of the body will be url encoded (which is the case when an HTML form is sent in the body).

If you want to know more about url encoding, read HTML URL Encoding Reference. You can also read more about the Express urlencoded() function.

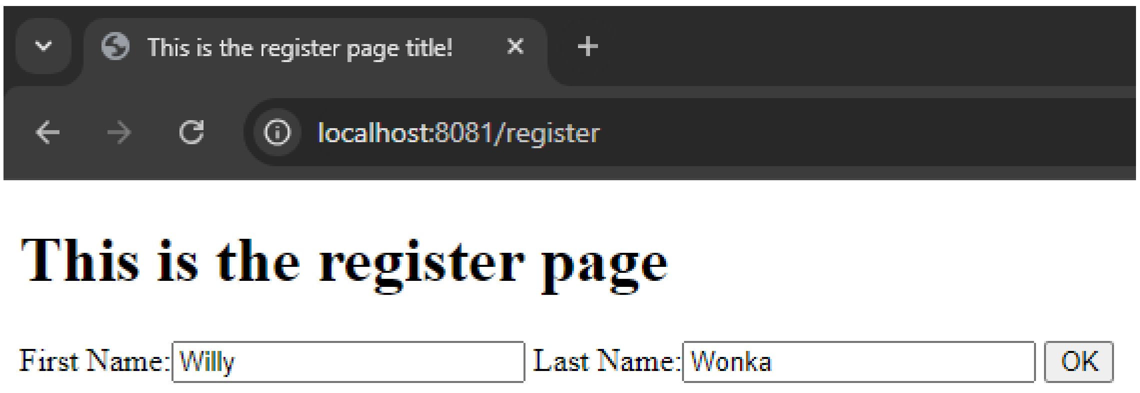
Task 5: Run the code

8. Run the `index.js` file with the following Terminal command:

```
node index.js
```

The above command will start a web server running on port 8081.

9. Browse to the register page at http://localhost:8081/register and enter a name:

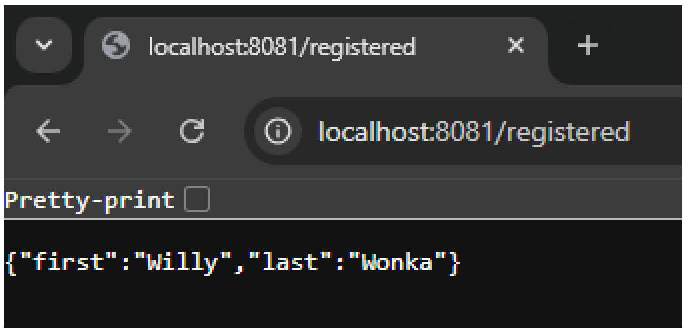

This is the register page title!

localhost:8081/register

This is the register page

First Name: Willy Last Name: Wonka

10. Press ‘OK’ to send the POST request to your web server. Your /registered route handler will process the request and return the response:



```
{"first":"Willy","last":"Wonka"}
```

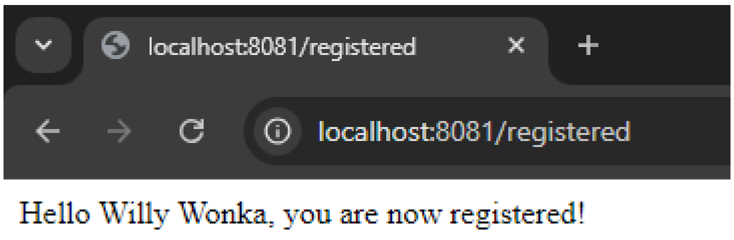
Task 6: Extract the individual fields from the body

The above response doesn’t look great. Let’s create a more nicely formatted response by extracting the individual fields.

11. Change the `/registered` route handler as follows:

```
router.post("/registered", (req,res) => {
  res.send("Hello "+ req.body.first + " "+ req.body.last +", you are now
registered!");
});
```

12. Run your application again, browse to the form and submit it. You should get a response like this:

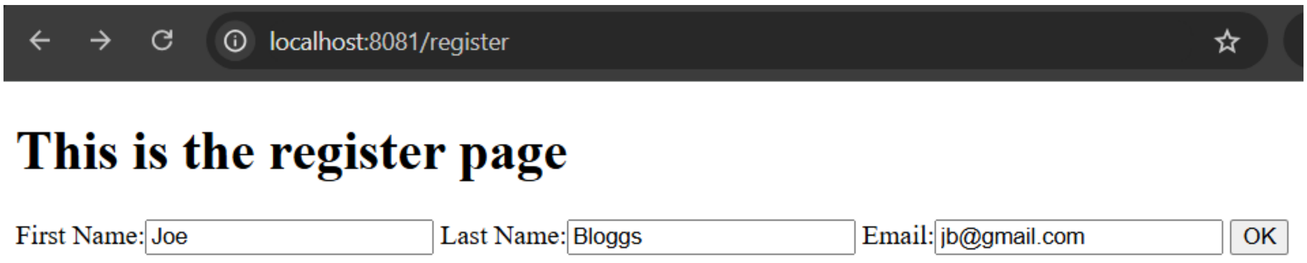


Hello Willy Wonka, you are now registered!

Task 7: Explore further

When tackling these lab activities, it’s always good to stretch yourself by doing some research and attempting some changes on your own.

Can you add an extra field ‘email’ to the register form? Your new register form should look like this:

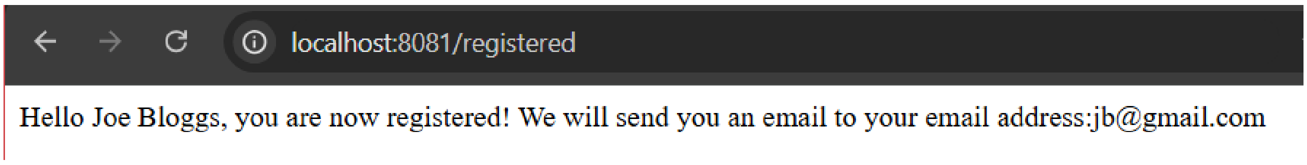


localhost:8081/register

This is the register page

First Name: Joe Last Name: Bloggs Email: jb@gmail.com

When you click on the 'Submit' button you should get the following output:



localhost:8081/registered

Hello Joe Bloggs, you are now registered! We will send you an email to your email address:jb@gmail.com

End of lab

Congratulations on completing this lab.

You have successfully created a web application that submits data in a HTML form. Furthermore, you are now able to collect that data using the POST request method.

If these links are broken, please let us know via the Student Portal.