Using GET and POST

In this lesson, we will learn the usage of GET and POST.

Understanding usage of *GET* and *POST*

Sending the form data to the server like we did in Exercise 4-4 has a big issue; it's privacy. The content of the form is sent as part of the URL, which can be copied from the address line of the browser.

For example, when you copy the request belonging to the last image in the previous lesson, you'll see this:

```
http://localhost:8467/FormProcessor.aspx
  ?salutation=mrval
  &fname=Istvan
  &lname=Novak
  &email=myemail%40mycompany.hu
  &membership=silver
  &login=istvan
  &pwd=secret
  &pwd2=secret
  &dev=on&csharp=on
  &comments=Surprise+me%21
```

> 📃**NOTE**: In the real URL there are no line breaks and indentations. Here I used them only for the sake of readability.

Here you can see my password strings! Even if you would use HTTPS, these passwords were part of the URL, so anyone could read it! HTTP and HTTPS provide a way to solve this issue. When you send a request to a server, you

can send it with either the GET or the POST verb, and each has its own semantic.
When sending form data, the browser uses the GET verb by default. The HTML specification says that GET should be used when the form “has no lasting observable effect on the state of the world”.

This is the plight when you send query parameters to the server that does not change anything in a database but only retrieves query results.

The POST verb should be used if the request sent to the server carries out some permanent change. The conference registration in an example of where you should use POST, because your intention is to save a registration and it is definitely a change in the application’s state.

When you use the GET verb, form data is passed in the URL just as you saw in the code snippet earlier. This means it can be seen by anyone. When the POST verb is used, the form data is sent in the body of the message and it cannot be seen in the browser’s address bar. Of course, the server-side also has to be prepared to extract the form data either from the message URL or from the message body, depending on whether GET or POST was used. You can specify the verb to use when sending form data with the method attribute of the `<form>` element:

```
<form action="formprocessor.html" method="post">
```

As you remember, the **formprocessor.html** page in Exercise-04-04 uses client-side JavaScript to display form parameters. If you use the POST method, the parameters posted to the server won’t be displayed by the JavaScript code.

There are many important and subtle differences between GET and POST in regard to web forms, and you’ll definitely need to learn more about them when you’re about to create web applications using heavy logic on the server-side.

Key points to remember

Now the most important things to remember are:

1. GET verbs transfer the form information in the URL making it visible to

users. Use GET only when you send query parameters to the server and when those parameters do not contain any sensitive data.

2. POST verbs transfer the form data in the request body, so it is not directly visible for users; however, it can easily be read with simple monitoring tools often built in the browsers. Use POST when sending information to the server that will trigger data changes.
3. If you have sensitive data in your web forms, use HTTPS with POST. In these exercises, you learned the basics of creating web forms that use text, dropdown lists, and options represented by radio buttons and checkboxes. It is time to take a look at other controls.

Great, now that we have covered GET and POST, we can move onto more form controls in HTML in the *next lesson*.