

COMP4021
Internet Computing

Posting Form Data

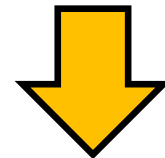
Gibson Lam

Using the GET Method

- In previous discussions, we looked at using the GET method to send HTML form data to the server
- Form data is sent using a query string put at the end of the URL, e.g.:

Firstname:

Lastname:



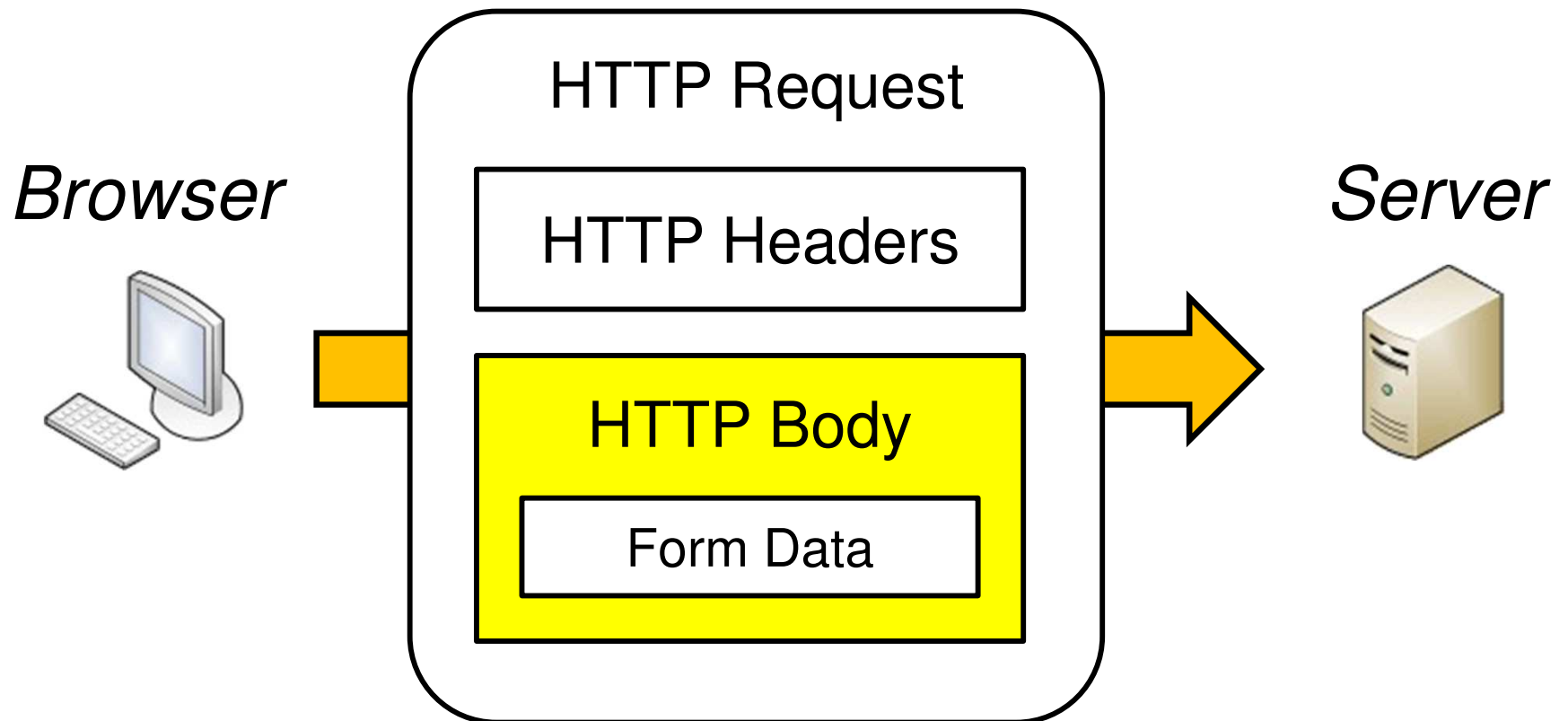
Press the submit button

`.../target?firstname=Gibson&lastname=Lam`

An example query string

Using the POST Method

- The POST method works differently
- Form data is sent to the server as part of the HTTP body



The HTML Form Example

- Here is the same form example with the form method changed to using POST:

Firstname:

Lastname:

The display of the form is the same as before

`<form method="post" action="/target">`
 `<p>Firstname:`
 `<input type="text" name="firstname"></p>`
 `<p>Lastname:`
 `<input type="text" name="lastname"></p>`
 `<p><input type="submit"></p>`
`</form>`

Posting the Form

Firstname:

Lastname:

- When sending the form using the POST method, the query string is put inside the HTTP body, like this:

```
POST /target HTTP/1.1
```

```
...
```

```
Content-Length: 29
```

```
Content-Type: application/x-www-form-urlencoded
```

```
...
```

*The length of
the form data*

```
firstname=Gibson&lastname=Lam
```

*The content
type of the
form data*

An example query string

Getting POST Data in Express

- You can read the query parameters in an Express server from forms that use the POST method
- To do that, you need to use the `urlencoded` middleware, as shown below:

```
app.use(  
  express.urlencoded({extended: true})  
);
```

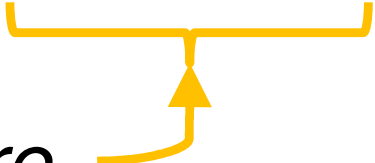
*Using an 'extended'
handler for the data*

Getting the Query Parameters

- After using the middleware, the query parameters are then available as a JavaScript object from `req.body`
- The following example can get the form data sent from the previous HTML form:

```
app.post("/target", (req, res) => {  
  const { firstname, lastname } = req.body;  
  ...  
});
```

The form data is stored here



Handling File Uploads

- Handling file upload in a form is quite different from handling normal input fields
- The HTML form is set up differently so that file content, which can be large, is sent within an HTTP request
- On the server side, you need to use an external npm package as an Express app does not handle file uploads appropriately

The HTML Form

- Here is how you write the `<form>` tag when you need to upload files:

*The encoding type of the form;
must be set to multipart/form-data*



```
<form enctype="multipart/form-data"  
method="post" action="/upload">
```

The diagram shows the HTML form tag with two yellow brackets. The top bracket spans the entire tag, and the bottom bracket highlights the `method="post"` attribute.

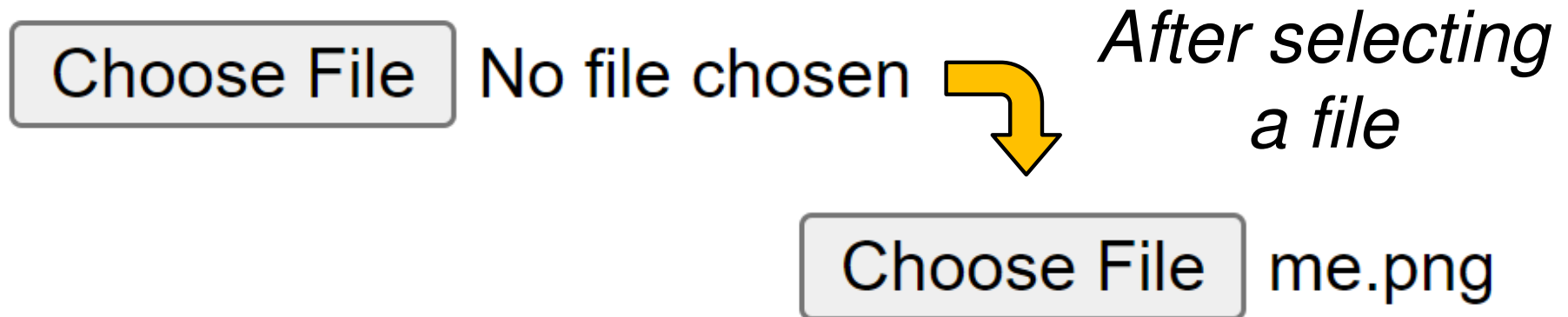
*You must use the
POST HTTP method*

The File Upload Input Fields

- A file upload input field is a form element, which is written like this:

```
<input type="file" name="myimage">
```

- The element contains a button and the display of the selected file so that the user can browse for the required file by clicking on the button:



The Encoding Type

- The encoding type to send the form data is multipart/form-data
- It is a MIME type which allows multiple parts of content to be transmitted in the same output
- On the right is the arrangement to put together N content parts

--Separator

Part 1 - Headers

Part 1 - Content

--Separator

Part 2 - Headers

Part 2 - Content

⋮

--Separator

Part N - Headers

Part N - Content

--Separator--

An Example Multi-Part Request

- Here is an HTTP POST request containing a file:

```
POST /upload HTTP/1.1
```

```
...some HTTP headers...
```

```
Content-Type: multipart/form-data;  
              boundary=Boundary-1234567890
```

```
--Boundary-1234567890
```

```
Content-Type: image/png
```

```
Content-Disposition: form-data;  
                    filename="me.png";  
                    name="myimage"
```

```
...content of the file...
```

```
--Boundary-1234567890--
```

*The part
contains
the file
content*

Handling Multi-Part in Express

- The Express package does not include functions for handling multi-part form data
- You need to install external packages to do that, such as the `multer` package
- You can install `multer` using `npm`, i.e.:


```
C:\Users\Gibson>npm install multer
```

Processing a File in Express

- In your Express server, you first need to create the middleware, like this:

```
const multer = require("multer");  
const upload = multer({ dest: "uploads/" });
```

- However, you don't usually use the middleware through the Express app



The folder you want to put the uploaded files

Using Multer in a Path

- In your server path, you can use multer if there is any file uploaded to the path, e.g.:

```
app.post("/upload",  
  Use multer for this path only → upload.single( "myimage" ),  
  (req, res) => {  
    ... Work with the file using req.file...  
  }  
);
```

The name of the form data containing the file

The Uploaded File

- You get the information of the uploaded file using `req.file`
- Below shows an example content of it:

```
{  
  fieldname: 'myimage',  
  originalname: 'me.png',  
  encoding: '7bit',  
  mimetype: 'image/png',  
  destination: 'uploads/',  
  filename: '3fe1d8b16093c246fc5814f70aa74ba6',  
  path: 'uploads\\3fe1d8b16093c246fc5814f70aa74ba6',  
  size: 269795  
}
```

*You can get the
file from here*




Copying the Uploaded File

- After receiving the file, you can choose to copy it somewhere, for example:

*This example
copies the file to
the 'public' folder*

```
fs.copyFile(req.file.path,  
            "public/" +  
            req.file.originalname);
```



- And delete the temporary file afterwards:

```
fs.unlink(req.file.path);
```