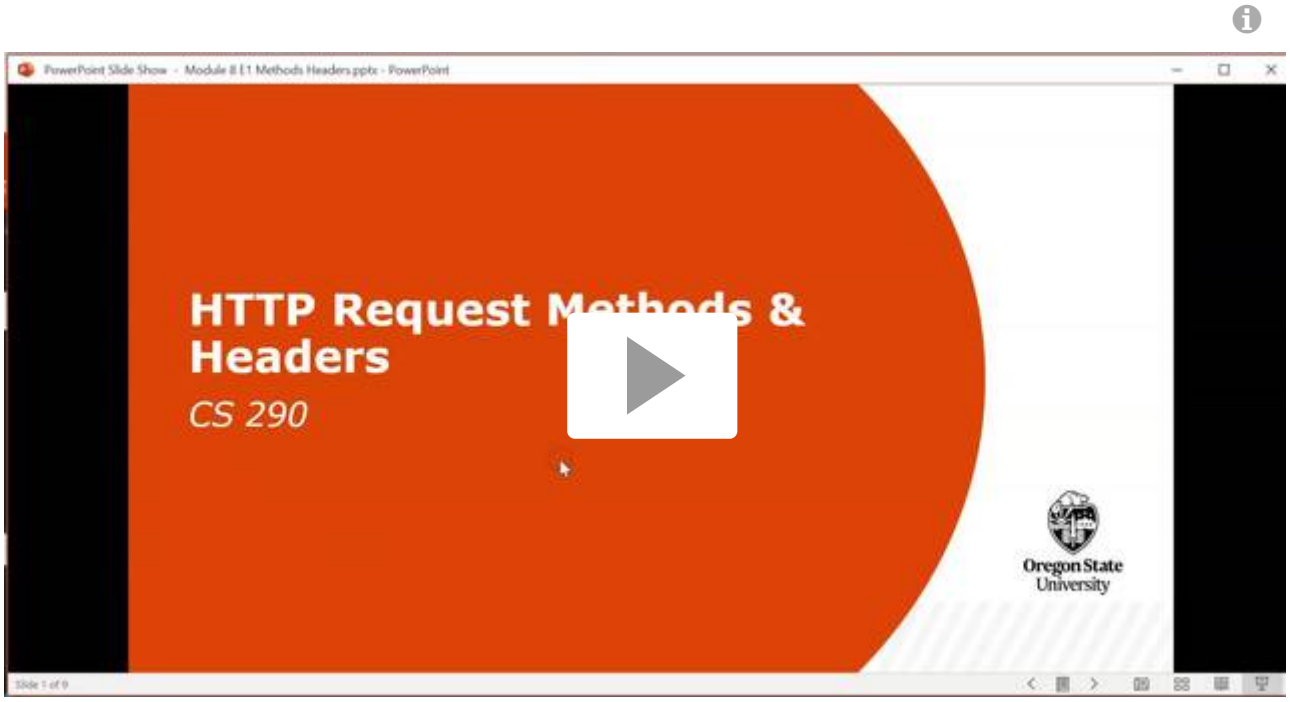
Exploration — HTTP Methods and Headers

Introduction

Exploration: HTTP Methods and Headers

In Module 1, we took a look at the HTTP protocol. In particular, we examined the different parts of HTTP requests and HTTP responses. In this exploration, we go into more details of certain important aspects of HTTP requests and responses that are important for developers of web applications. We will first take a look at HTTP request methods and then study some important HTTP headers.



0:00 / 23:28

HTTP Request Methods

We have looked at examples of using `GET` and `POST` methods in HTTP requests. We again look at these methods to identify prescribed use cases for these methods. While `GET` and `POST` are the HTTP methods used most widely, we will also identify a few other HTTP methods that are also frequently used for specific purposes.

GET

When we type a URL in a browser, the browser sends an HTTP `GET` request to the server. `GET` requests should be used to read data from the server. The `GET` method is thus analogous to the **Read/Retrieve** operation among the CRUD operations. With a `GET` request, information is passed to the server via the *URL* path and the *query* string. While the HTTP specification does not prohibit sending a body with an HTTP `GET` request, sending a *body* with a `GET` request is *not recommended* at all.

POST

The HTTP `POST` method sends data to the server in the *body* of the request. Many times `POST` is used when submitting **forms**. A prescribed use of the `POST` method is for creating a *new* resource, e.g., a new document in a database. This means if an HTTP request being sent to the server to perform a **Create** operation, then the `POST` method should be used.;

PUT

The HTTP `PUT` method is used for **Update** operations. A prescribed use of the `PUT` method is for HTTP requests in which a resource is completely *replaced* by the data in the HTTP request with the `PUT` method. For example, if we wanted to send an HTTP request to replace all the properties of a document to new values, then it would be an appropriate use case for the `PUT` method. Data is sent in the *body* of the request for HTTP requests that use the `PUT` method.

PATCH

The prescribed use of the HTTP `PATCH` method is for **partial Updates** of a resource, unlike `PUT` which is used for completely replacing the resource. For example, if we want to send an HTTP request to update some, but not all, properties of a document, then it would be an appropriate use case for the `PATCH` method. Support for the `PATCH` method is *not universal,* and many web server support partial updates using the `PUT` method.

DELETE

The prescribed use of the HTTP `DELETE` method is to **Delete** a resource. For example, if we want to send an HTTP request to delete a document, then it would be an appropriate use case for the `DELETE` method.

HEAD

Unlike the methods we discussed above, the `HEAD` method is *not* related to CRUD operations. The `HEAD` method is similar to `GET` in that it requests a resource for retrieval. However, the response to a request using the `HEAD` method does not include the resource, but *only includes the status line*

and the *HTTP **response headers***. This is used by clients, such as browsers, to determine if the resource in their **cache** is still fresh, or should they now issue a `GET` request for the resource.

HTTP Methods and Express Routes

The term **endpoint** is used for the combination of a URL and an HTTP method. Two requests that have the same URL, but different HTTP methods, are considered two different endpoints. In Express it is simple to define routes based on the combination of a URL and HTTP method.

- We have already used `app.get` and `app.post` to define routes for HTTP `GET` and HTTP `POST` methods, respectively.
- The Express API provides `app.put`, `app.patch` and `app.delete` to define routes for HTTP methods `PUT`, `PATCH` and `DELETE`.
- The first parameters for all of these functions, which are collectively referred to as `app.METHOD` functions, is the `path` or URL.
- For the same URL, we can define **multiple route handlers** based on the HTTP method by adding routes that have the same value of the argument `path`, but use a different HTTP method.
- Express also provides a function `app.all` which handles requests based on the `path` arguments, regardless of the HTTP method.

Example: HTTP Methods and Express Routes

In the previous module, we implemented a controller for CRUD operations on a movie database. In that controller, we defined 4 routes for the CRUD operations by specifying 4 different values for the `path` argument, i.e., `/create`, `/retrieve`, `/update`, `/delete`, but using the same HTTP method `GET`. Here is a different controller with 4 routes that all have the same value for the path argument, i.e., `movies`, but use the HTTP method to differentiate between the 4 types of requests the controller handles. This type of controller that uses the HTTP method for defining routes forms the basis of **REST API** which we will study in a later exploration in this module.

```
app.post('/movies', (req, res) => {
  // Code for create operation
}

app.get('/movies', (req, res) => {
  // Code for read/retrieve operation
}

app.put('/movies', (req, res) => {
  // Code for update operation
}

app.delete('/movies', (req, res) => {
  // Code for delete operation
}
```

HTTP Headers

HTTP headers (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) are used by the client and the server to pass additional information about the HTTP request or HTTP response.

- The headers consist of *name:value* pairs separated by a `:`.
- The header names are *case-insensitive*.
- There are some headers that are only used in requests, others that are only used in responses, and some are used in either.

In an earlier Exploration, we briefly discussed some headers. We now take a deeper look at some important headers that convey information about the format of the data being requested by the client, or being sent by either the client or the server. This requires us to understand the concept of MIME types, which we discuss next.

MIME Types

When an HTTP *request* or *response* contains a body, the server needs to know *how to interpret* the bytes sent by the client. This information is conveyed by using values from a standard called **Multipurpose Internet Mail Extensions (MIME) type** (https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types). MIME types are also called media types or Internet Media Types

- The typical structure of a MIME type is of the form `type/subtype`, i.e., two strings separated by a `/`.
- The type indicates a general category.
  - Example types include `text` or `application` and `image`.
- The subtype indicates the exact type of data within that category.
  - For the `text` type, subtypes include:
    - `css` for CSS,
    - `csv` for comma separated values,
    - `javascript` for JavaScript code,
    - `html` for HTML,
    - `plain` for plain text.
  - For the `image` type, subtypes include `jpeg`, `png`.
  - For the `application` type, subtypes include:
    - `json` for JSON data,
    - `pdf` for PDF documents,
    - `xml` for XML data,
    - `zip` for ZIP files.

Headers with MIME Types

Accept Header

The `Accept` header is used in HTTP requests by a client to tell the server about the types of data the client can handle. In general, the value of the `Accept` header is one or more MIME types separated by commas.

- The value `*/*` means that the client can handle all types of data from the server.
- The value `type/*` means that the client can accept all subtypes of the `type`.
  - For example, `text/*` means that the client can handler all subtypes of the `text` type.
- If the client is capable of handling multiple specific MIME types, it can tell the server about this by sending these MIME types in the `Accept` header separated by commas.
  - For example, `Accept : application/json, text/html` indicates that the client can handle HTML and JSON documents.

Content-Type Header

The `Content-type` header can be used in both HTTP requests and HTTP responses. HTTP requests that contain a body, e.g., a `POST` or a `PUT` request, include this header to tell the server the MIME type of the body. Similarly, HTTP responses with a body include the `Content-type` header to tell the client the MIME type of the body in the response.

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

The most common MIME type for `POST` bodies is `application/x-www-form-urlencoded` which is used when we submit a form with method set to Post: `<form method="POST">`. This is also referred to as HTML form encoding or URL form encoding.

- In this MIME type, the form data is encoded as key-value pairs.
- The key and the value in a pair are separated by a `=`, while different key-value pairs are separated by `&`.
- Any non-alphanumeric characters in keys or in values are URL encoded.

Summary

In this exploration, we studied the most commonly used HTTP request methods. We then looked at MIME types and their use in `Accept` and `Content-Type` headers.

Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- **Basics of HTTP (https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP)** at MDN provides a good description of the HTTP protocol.
- A description of **HTTP methods (https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)** at MDN.

- The specification of HTTP request methods is given in **Section 4 of RFC 7231** (https://datatracker.ietf.org/doc/html/rfc7231#section-4), except PATCH which is specified in **RFC 5789** (https://datatracker.ietf.org/doc/html/rfc5789)
- Details of many **HTTP headers** (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) can be found at MDN.
- **MIME types** (https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types) described at MDN and the official listing of MIME types at **IANA** (https://www.iana.org/assignments/media-types/media-types.xhtml).