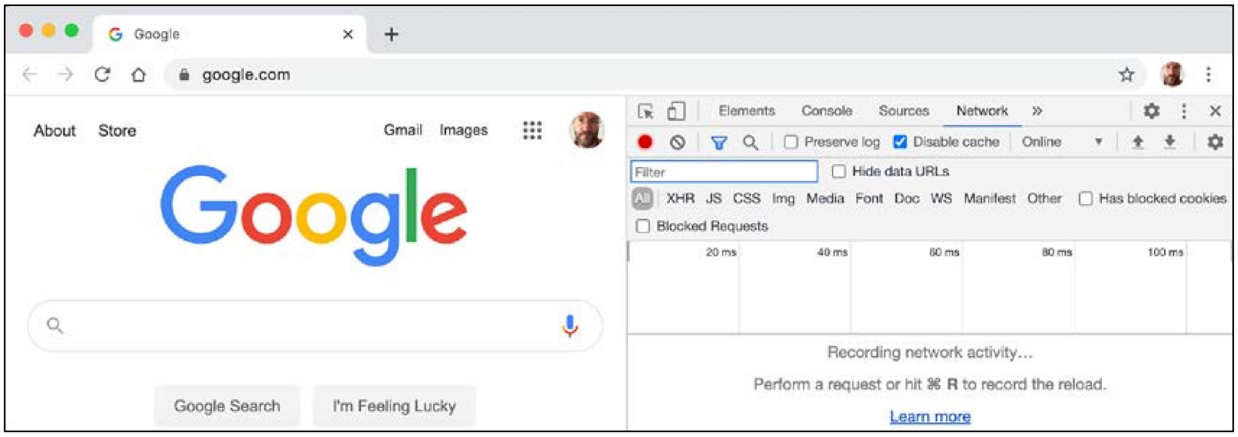
3. Click on the **Network** tab, and Chrome should immediately start recording the network traffic between your browser and any web servers, as shown in the following screenshot:

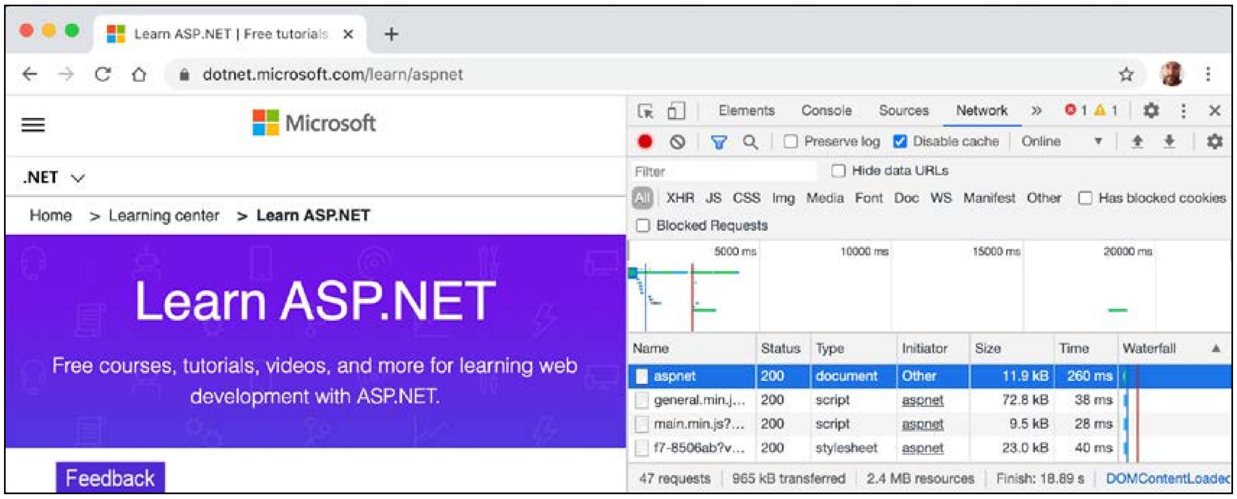


Recording network activity…

Perform a request or hit ⌘ R to record the reload.

Figure 15.2: Chrome recording network traffic

4. In Chrome's address box, enter the following URL: `https://dotnet.microsoft.com/learn/aspnet`.
5. In the **Developer tools** window, in the list of recorded requests, scroll to the top and click on the first entry, the **document**, as shown in the following screenshot:

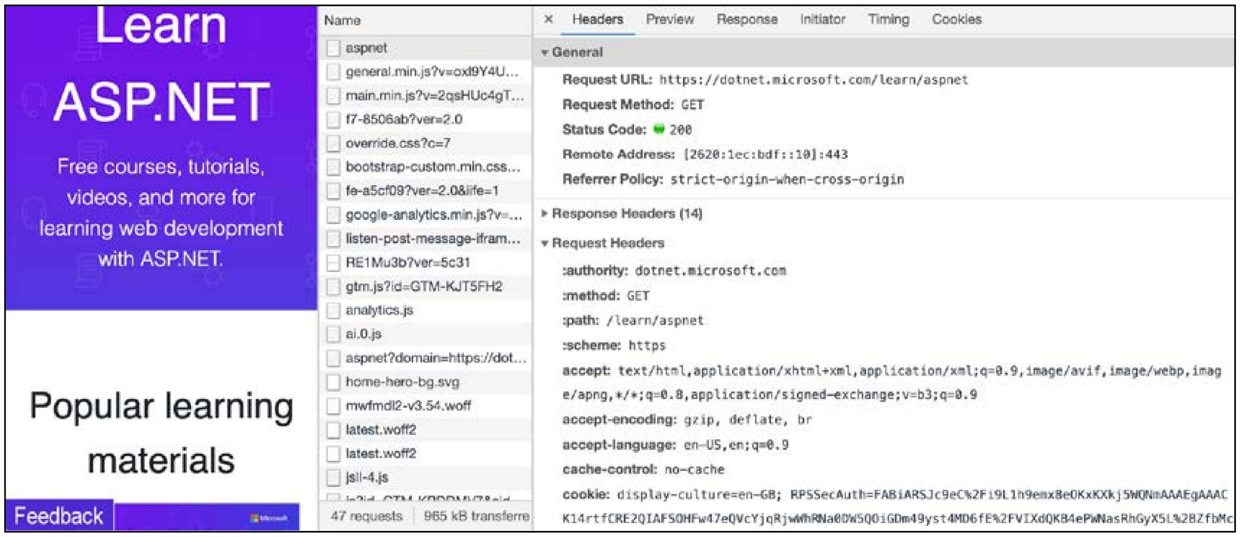


Name	Status	Type	Initiator	Size	Time	Waterfall
aspnet	200	document	Other	11.9 kB	260 ms	
general.min.j…	200	script	aspnet	72.8 kB	38 ms	
main.min.js?…	200	script	aspnet	9.5 kB	28 ms	
f7-8506ab?v…	200	stylesheet	aspnet	23.0 kB	40 ms	

47 requests | 965 kB transferred | 2.4 MB resources | Finish: 18.89 s | DOMContentLoaded

Figure 15.3: Recorded requests

6. On the right-hand side, click on the **Headers** tab, and you will see details about the request and the response, as shown in the following screenshot:


Learn
ASP.NET
Free courses, tutorials, videos, and more for learning web development with ASP.NET.
Popular learning materials
Feedback
Name
Headers
Preview
Response
Initiator
Timing
Cookies

```
aspnet
general.min.js?v=oxl9Y4U...
main.min.js?v=2qsHUc4gT...
f7-8506ab?ver=2.0
override.css?c=7
bootstrap-custom.min.css...
fe-a5cf09?ver=2.0&life=1
google-analytics.min.js?v=...
listen-post-message-ifram...
RE1Mu3b?ver=5c31
gtm.js?id=GTM-KJT5FH2
analytics.js
ai.0.js
aspnet?domain=https://dot...
home-hero-bg.svg
mwfmdl2-v3.54.woff
latest.woff2
latest.woff2
jsll-4.js
47 requests | 965 kB transferre
```

```
▾ General
    Request URL: https://dotnet.microsoft.com/learn/aspnet
    Request Method: GET
    Status Code: ● 200
    Remote Address: [2620:1ec:bdf::10]:443
    Referrer Policy: strict-origin-when-cross-origin
▸ Response Headers (14)
▾ Request Headers
    :authority: dotnet.microsoft.com
    :method: GET
    :path: /learn/aspnet
    :scheme: https
    accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
    e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
    accept-encoding: gzip, deflate, br
    accept-language: en-US,en;q=0.9
    cache-control: no-cache
    cookie: display-culture=en-GB; RPSSecAuth=FABiARSJc9eC%2Fi9L1h9emx8eOKxKXkj5WQNmAAAEgAAAC
    K14rtfCRE2QIAFSOHFw47eQVcYjqRjwWhRNa0DW5Q0iGDm49yst4MD6fE%2FVIXdQKB4ePWNasRhGyX5L%2BZfbMc
```

Figure 15.4: Request and response details

Note the following aspects:

- **Request Method** is `GET`. Other methods that HTTP defines include `POST`, `PUT`, `DELETE`, `HEAD`, and `PATCH`.
- **Status Code** is `200 OK`. This means that the server found the resource that the browser requested and has returned it in the body of the response. Other status codes that you might see in response to a `GET` request include `301 Moved Permanently`, `400 Bad Request`, `401 Unauthorized`, and `404 Not Found`.
- **Request Headers** sent by the browser to the web server include:
    - **accept**, which lists what formats the browser accepts. In this case, the browser is saying it understands HTML, XHTML, XML, and some image formats, but it will accept all other files `*/*`. Default weightings, also known as quality values, are 1.0. XML is specified with a quality value of 0.9 so it is preferred less than HTML or XHTML. All other file types are given a quality value of 0.8 so are least preferred.
    - **accept-encoding**, which lists what compression algorithms the browser understands. In this case, GZIP, DEFLATE, and Brotli.
    - **accept-language**, which lists the human languages it would prefer the content to use. In this case, US English, which has a default quality value of 1.0, and then any dialect of English that has an explicitly specified quality value of 0.9.
- **Response Headers**, **content-encoding** tells me the server has sent back the HTML web page response compressed using the GZIP algorithm because it knows that the client can decompress that format.

7. Close Chrome.

Client-side web development

When building websites, a developer needs to know more than just C# and .NET Core. On the client (that is, in the web browser), you will use a combination of the following technologies:

- **HTML5**: This is used for the content and structure of a web page.
- **CSS3**: This is used for the styles applied to elements on the web page.
- **JavaScript**: This is used to code any business logic needed on the web page, for example, validating form input or making calls to a web service to fetch more data needed by the web page.

Although HTML5, CSS3, and JavaScript are the fundamental components of frontend web development, there are many additional technologies that can make frontend web development more productive, including Bootstrap, the world's most popular frontend open source toolkit, and CSS preprocessors like SASS and LESS for styling, Microsoft's TypeScript language for writing more robust code, and JavaScript libraries like jQuery, Angular, React, and Vue. All these higher-level technologies ultimately translate or compile to the underlying three core technologies, so they work across all modern browsers.

As part of the build and deploy process, you will likely use technologies like Node.js; **Node Package Manager** (**NPM**) and Yarn, which are both client-side package managers; and Webpack, which is a popular module bundler, a tool for compiling, transforming, and bundling website source files.

More Information: This book is about C# and .NET Core, so we will cover some of the basics of frontend web development, but for more detail, try *HTML5 and CSS3: Building Responsive Websites*, at: `https://www.packtpub.com/product/html5-and-css3-building-responsive-websites/9781787124813h`

Understanding ASP.NET Core

Microsoft ASP.NET Core is part of a history of Microsoft technologies used to build websites and web services that have evolved over the years:

- **Active Server Pages** (**ASP**) was released in 1996 and was Microsoft's first attempt at a platform for dynamic server-side execution of website code. ASP files contain a mix of HTML and code that executes on the server written in the VBScript language.
- **ASP.NET Web Forms** was released in 2002 with the .NET Framework, and is designed to enable non-web developers, such as those familiar with Visual Basic, to quickly create websites by dragging and dropping visual components and writing event-driven code in Visual Basic or C#. Web Forms can only be hosted on Windows, but it is still used today in products such as Microsoft SharePoint. It should be avoided for new web projects in favor of ASP.NET Core.