WEB524

WEB PROGRAMMING ON WINDOWS

WEEK 10 – LECTURE 1

NON-TEXT MEDIA TYPES

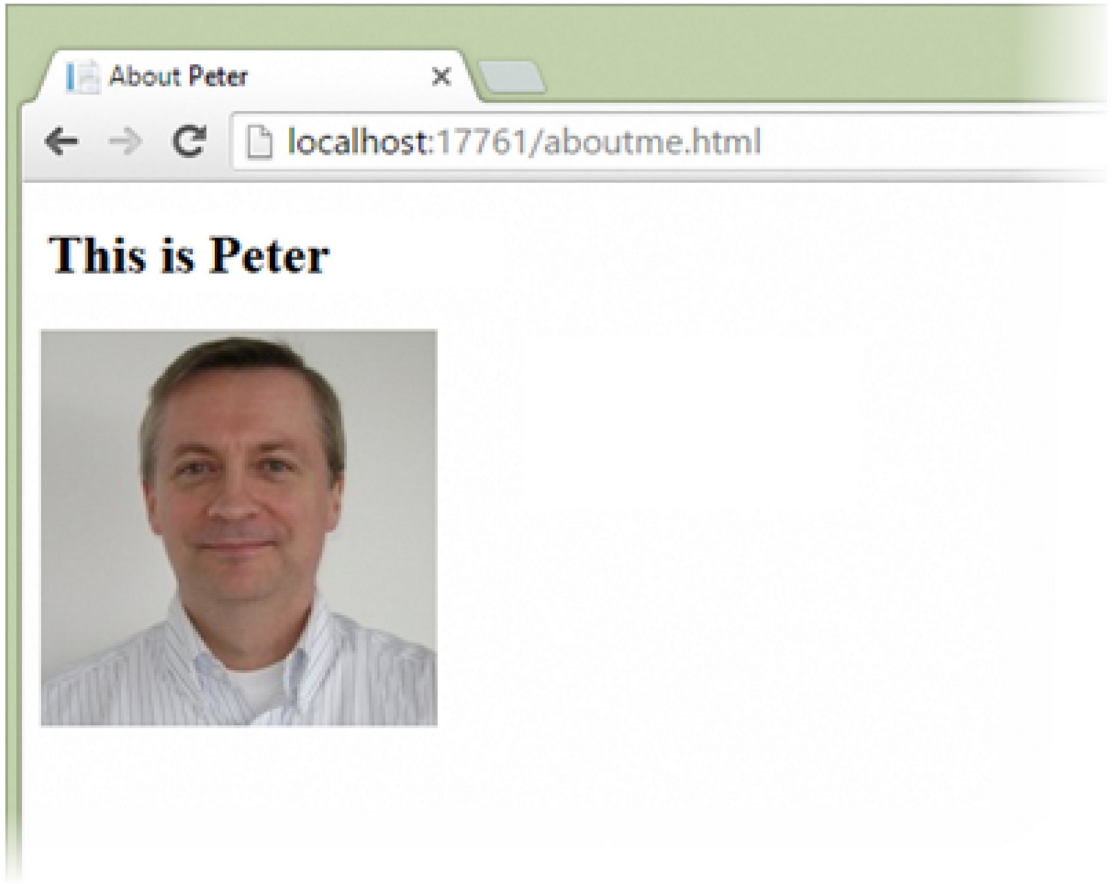
This Week

- We will work with non-text media types.
- Learn to develop web apps that capture, store, and deliver these content types.
- Today, we'll cover techniques for handling a simple scenario where an entity class (Product, Player, Vehicle, Album, etc.) includes a single photo.
- Next class, we'll covers techniques for handling a more complex scenario, where an entity class is associated with a collection of non-text media items.

Images in HTML

- Every student in this course has coded a static HTML document that includes a photo.

```
<!DOCTYPE html>
<html>
  <head>
    <title>About Peter</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h2>This is Peter</h2>
    <img src="peter.jpg" alt="" />
  </body>
</html>
```



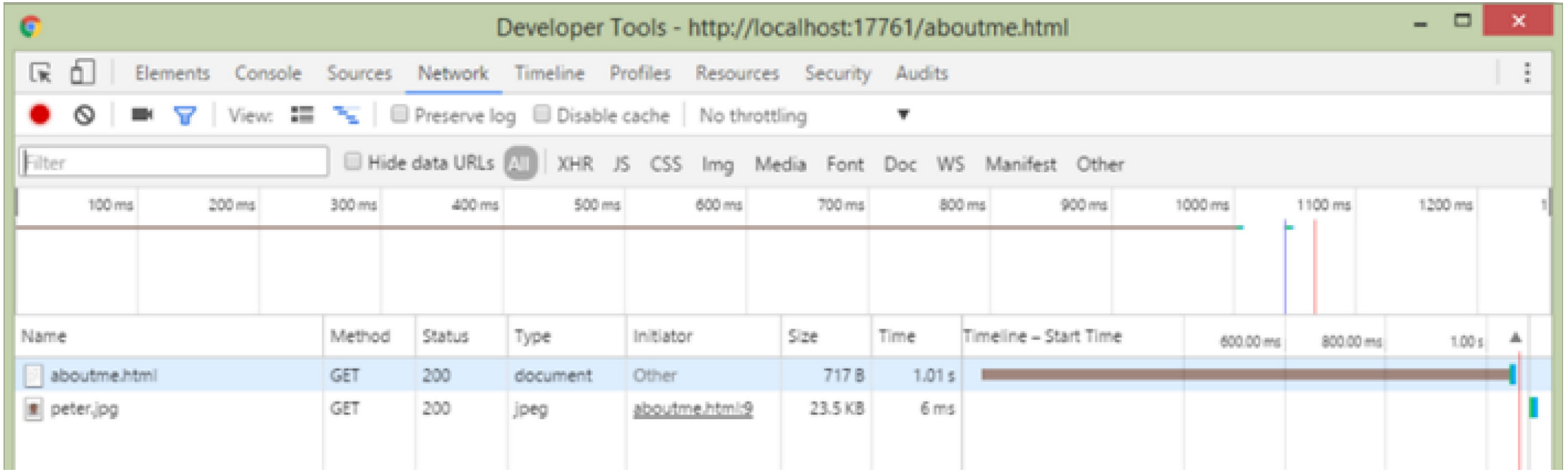
The <img> Element

- Notice that the photo is defined by the HTML <img> element.
- The <img> element has two attributes that must be included, **alt** and **src**.
- The value of the **src** attribute must be an absolute or relative path to the photo.
- In other words, the value of the **src** attribute is a **URL**.

```
<!DOCTYPE html>
<html>
  <head>
    <title>About Peter</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h2>This is Peter</h2>
    <img src="peter.jpg" alt="" />
  </body>
</html>
```

Fetching a Document with a Browser

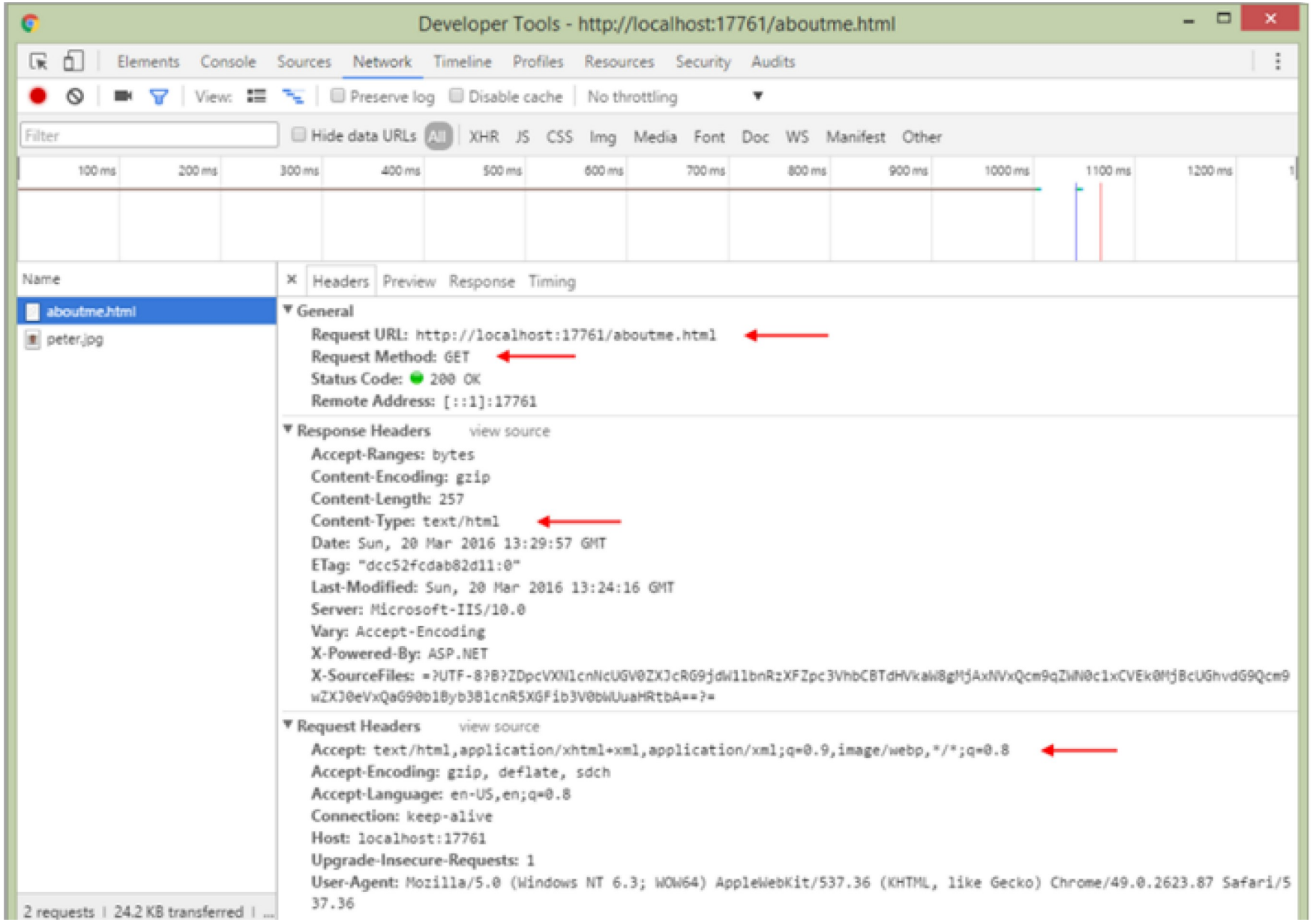
- To see how a browser will fetch and render a document, use the browser developer tools.
- For example, in Chrome:
  - Open the developer tools
  - Select the Network item
  - Request the document
  - … This is what you'll see



- Notice that there were TWO requests:
  1. The first request was for “aboutme.html”
  2. The second request was for “peter.jpg”

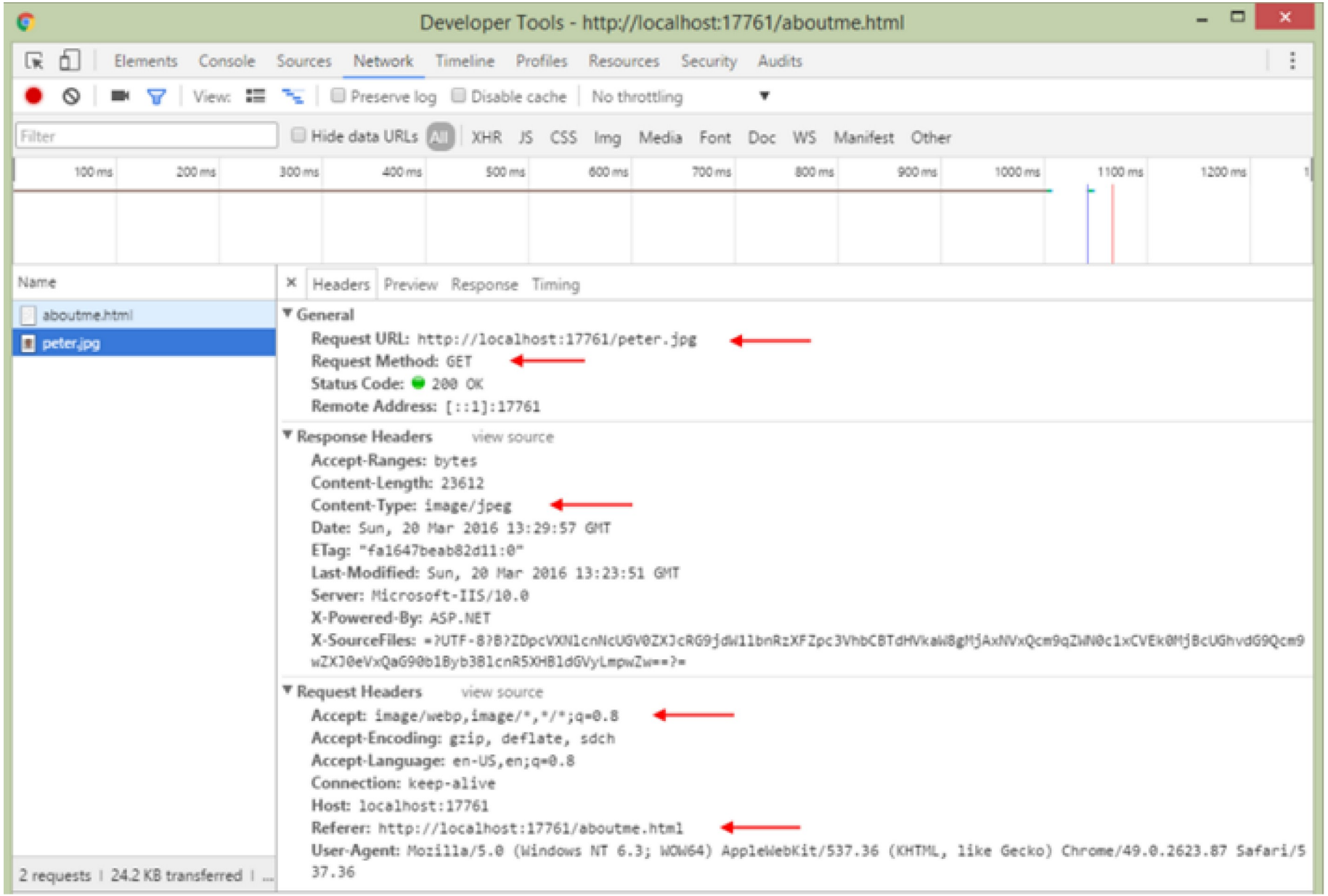
Fetching a Document with a Browser

- When the browser fetches an HTML document, it parses the content.
- When it finds an HTML **<img>** element, it sends another request to the URL (in the value of the **src** attribute).
- Here's what the request for "aboutme.html" looks like, notice:
  - The request URL and method
  - The "Accept" request header, in which the requestor indicates the acceptable content types
  - The "Content-Type" response header, in which the server identifies the resource's content type


Developer Tools - http://localhost:17761/aboutme.html
Elements Console Sources Network Timeline Profiles Resources Security Audits
Preserve log Disable cache No throttling
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other
100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1000 ms 1100 ms 1200 ms
Name
aboutme.html
peter.jpg
Headers Preview Response Timing
General
Request URL: http://localhost:17761/aboutme.html
Request Method: GET
Status Code: 200 OK
Remote Address: [::1]:17761
Response Headers view source
Accept-Ranges: bytes
Content-Encoding: gzip
Content-Length: 257
Content-Type: text/html
Date: Sun, 20 Mar 2016 13:29:57 GMT
ETag: "dcc52fcdab82d11:0"
Last-Modified: Sun, 20 Mar 2016 13:24:16 GMT
Server: Microsoft-IIS/10.0
Vary: Accept-Encoding
X-Powered-By: ASP.NET
X-SourceFiles: =?UTF-8?B?ZDpcVXNlcnNcUGV0ZXJcRG9jdW1lbnRzXFZpc3VhbCBTdHVkaW8gMjAxNVxQcm9qZWN0c1xCVEk0MjBcUGhvdG9Qcm9wZXJ0eVxQaG90b1Byb3B1cnR5XGFib3V0bWUuaHRtbA==?=
Request Headers view source
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Connection: keep-alive
Host: localhost:17761
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36
2 requests | 24.2 KB transferred

Fetching an Image with a Browser

- Here's what the request for "peter.jpg" looks like, notice:
  - The request URL and method.
  - The "Referrer" request header.
  - The "Accept" request header, in which the requestor indicates the acceptable content types.
  - The "Content-Type" response header, in which the server identifies the resource's content type.


Developer Tools - http://localhost:17761/aboutme.html
Elements Console Sources Network Timeline Profiles Resources Security Audits
View: Preserve log Disable cache No throttling
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other
100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1000 ms 1100 ms 1200 ms
Name
aboutme.html
peter.jpg
Headers Preview Response Timing
General
Request URL: http://localhost:17761/peter.jpg
Request Method: GET
Status Code: 200 OK
Remote Address: [::1]:17761
Response Headers view source
Accept-Ranges: bytes
Content-Length: 23612
Content-Type: image/jpeg
Date: Sun, 20 Mar 2016 13:29:57 GMT
ETag: "fa1647beab82d11:0"
Last-Modified: Sun, 20 Mar 2016 13:23:51 GMT
Server: Microsoft-IIS/10.0
X-Powered-By: ASP.NET
X-SourceFiles: =?UTF-8?B?ZDpcVXN1cnNcUGV0ZXJcRG9jdW1lbnRzXFZpc3VhbCBTdHVkaW8gMjAxNVxQcm9qZWN0c1xCVEk0MjBcUGhvdG9Qcm9wZXJ0eVxQaG90b1Byb3B1cnR5XHB1dGVyLmpwZw==?=
Request Headers view source
Accept: image/webp,image/*,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Connection: keep-alive
Host: localhost:17761
Referer: http://localhost:17761/aboutme.html
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36
2 requests | 24.2 KB transferred

Definitions (Media Type & Media Item)

- The terminology used to identify, define, and categorize web content is a bit sloppy.
- The example from earlier worked with two kinds of web content, text and a photo.
- Each kind of web content is defined by its **media type**. In its strict definition, every kind of content – text and non-text – has a media type.
- For example, an HTML document's media type is text/html.
- We can work with many kinds of non-text content, including audio, video, digital documents (e.g. PDF).
- In general, each non-text content item is defined as a **media item**.

Definitions (MIME)

Multipurpose Internet Mail Extensions was originally developed for organizing the structure of contents of the messages transferred over SMTP protocol. But it's adopted in other communication protocols, such as HTTP.
Now, MIME is a specification which describes how to show the structure, format and nature of some sort of data.

- While each media item can be defined by its media type (in its Content-Type header), web developers can also use an application of media types, known as MIME.
- Among other capabilities, the MIME standard describes how to deliver/render/present a media item.
- We often see this on pages that offer a PDF document:
  - Sometimes clicking the link to a PDF document will display the document in the browser.
  - Sometimes the PDF document will be downloaded (with or without a popup dialog that enables the user to continue and/or save to a specific location on their device).
  - This behavior is influenced by the web developer, by using the Content-Disposition header

Definition Summary

To repeat the definitions and terminology from the last two slides:

- We will generally refer to non-text content as **media items**.
- A **media item** is defined by its **media type**.
- https://stackoverflow.com/a/61607262

We Just Learned ...

- If an HTML document includes a media item, the browser creates a separate request for that media item.
- Therefore, whatever solution we create, it must expose the media item as a URL.

- Some students may have seen or used a “data URI”. The Data URI scheme provides a way to embed the media item’s encoded data as the value for the “src” attribute.
- We will not be covering this topic as it is beyond the scope of this course. You can read more here: https://en.wikipedia.org/wiki/Data_URI_scheme

Statically-Handled Media Items

- In the past, you have written web pages and web apps that have used static media items.
- In other words, you – the web developer – have added media items (photos etc.) to your project, alongside the HTML, CSS, and JavaScript source code files.
- Each had its own name (e.g. logo.png, banner1.jpg, etc.).
- If you had many similar media items, maybe you created a folder (e.g. /images/) to organize them within your project.
- Each media item could then be referenced by a URL, using an absolute or relative path from the document that includes the HTML <img> element.

Dynamically-Handled Media Items

- We can also dynamically handle a media item
- In an ASP.NET MVC web app, what this means is that the URL to the media item would be handled by an action in a controller.
- Why would we do this? For many reasons… including design, user experience, security, scale, and flexibility.
- In other words, we want to actively manage the way that media items are handled.

Storage of Media Items

- Media Items can be stored in:
  - **File system**: In the file system, within the URL namespace of the web app.
  - **Persistent store**: In a persistent store like a database. The media item can be an embedded property of another entity class or it can be defined in its own dedicated entity class.
  - **Hybrid combination**: A media item is in the file system, but its metadata is in the persistent store. Similar as above, the media item's metadata can be in embedded properties of another entity class, or it can be defined in its own dedicated entity class
- When the web app is hosted in the cloud (e.g. Azure), the cloud service may offer additional ways to store media items

Implications – File System Storage

- In a web app that uses the file system to store media items, it must have a folder to hold the media items, maybe called “images” or “assets”, or something that fits with the web app’s problem domain.
- In an ASP.NET MVC web app, you may be tempted to use an existing folder, but you shouldn’t.
- You cannot use App_Data because its contents are not publicly accessible.
- You shouldn’t use Content because that has a specific use now and may evolve in the future.
- Create your own folder.
- Using a static delivery approach, the file name extension of the media item is super important. The web server (that is hosting your web app) will use the extension to set the Content-Type header in the response.

Implications – File System Storage

- File names (for the media items) are important.
  - Their length must be reasonable.
  - They must be unique.
  - To simplify coding and handling, the character set used for the file names should be neutral, especially for a web app that's used worldwide.
- When using a file system storage approach, each media item is logically associated with an entity object in your problem domain.
- How do you maintain that association?
- It is likely that the media item itself cannot store information about the association and the file name may not be enough to do the job.
- Therefore, the entity object must include a property to hold the media item's file name.

Implications – File System Storage

- How should a file system media item be delivered to the browser?
- Statically, using a URL that maps to its file system location and file name? Or through some other dynamic and managed approach?
- Beyond file names, extension names and created/modified dates, the file system will not (and cannot) support metadata querying.
- It will become difficult (and non-performant) to handle a query, for example, for all photos larger than 1000px wide and 800px tall.
- It may be difficult or impossible to store descriptive metadata in the media item (depending upon its format).
- In a web app that uses both a persistent store (i.e. database) and the file system for data storage, the web app manager now must manage two separate storage locations, and ensure they're backed up and secure.

Implications – Persistent Store

- In a database-hosted data store, the media item's data is stored in a byte array (the C# type is Byte[]).
- This data type maps nicely to both the storage and delivery components involved.
- As a result, it's on equal footing with the file system in this respect.
- You MUST store the media item's media type string during the file upload task, that metadata is available in the Content-Type of the request.

Implications – Persistent Store

- An advantage of this storage approach is that media item metadata can be stored alongside its data, as additional properties in the class that holds the media item's data.
- For example, you can store a "Title", a lengthy "Description", and maybe some "tags" or "keywords".
- Other properties that are relevant to the media item can be stored, and their values can come from the user, or by programmatically inspecting the media item. (For example, the pixel resolution of an image can be extracted from the data, without user intervention.)

Implications – Persistent Store

- Delivery of the media item is managed by your app.
- Usually, a special-purpose controller accepts an identifier in the request URL, then locates the requested media item, and finally delivers it to the requestor.
- Backing up the web app's data is simple since there's a single location for all the app's data (the persistent store).
- A frequent and notable criticism of this approach is that a media item is transformed when stored, then again when retrieved from the data store.
- For large media items, this work can be considerable and may hurt performance.

Implications – Hybrid Approach

- It is also possible to combine both techniques.
- Your app stores the media item's data in the file system and metadata in the data store.
- The file system location can be a folder in the web app's URL namespace. Often, the file name is a GUID *(or an adaptation of a GUID)*.
- The data store class includes media item metadata including (most importantly) the file name.
- Like the persistent store approach, a special-purpose controller is used to help locate and deliver the requested media item.
- This approach is often used in situations where the media item size is large (multi-MB or multi-GB), or where there's a large number (e.g. millions) of media items to manage.
- As you would suspect, today's best-known and widely-used web apps use this approach. For example, YouTube, Facebook, Netflix, etc.

Entity Class Design Considerations

** Important **

- As noted earlier, when using the persistent store, an entity class that includes a media item MUST have two properties:
  1. A byte array, for the media item's data.
  2. A string for the media type.
- Beyond that, other properties (aka *metadata*) can be added if desired.

Entity Class Design Considerations

- If the design of an entity class is intended to hold one distinct and unique media item, then it is acceptable to simply add these properties to the entity class.
- For example, if an Employee class will hold one – *and only one* – photo for the employee's photo identification card, then the Employee class will include the two properties.

“… Base” View Model Class Considerations

- In the “…Base” view model classes, DO NOT include the two properties.
- In fact, NEVER include these properties in view models that are used to deliver data to a browser.
- If you want or need to detect the presence of a media item, then you can add a property to do this.
  - Perhaps a bool property that is programmatically-determined.
  - Or, an int property that holds the size of the byte array data.

Collections of Media Items

- If an entity object needs a collection of media items (e.g. a photo gallery for a Product object)
- Create a separate entity class for the media item and then configure the to-one and to-many (or whatever) associations needed to meet the needs of the problem domain.
- If the primary purpose of the app is to manage media items (like YouTube, for example), then the entity class(es) must be designed around the needs of the app and the media item entity classes become the central part of the design model. This topic is beyond the scope of this course.

Hands-On – A Solution to Capture, Store and Deliver Media Items

- Download the "**PhotoProperty**" code example and the document "**Lecture 10.1 Hands On Lab**" from Blackboard.
- Study the document and code example.

- **Answer the following questions:**
  - In the "Create" view for a Vehicle, why must we use a different constructor to render the form?
  - In the Manager.VehicleAdd() function, why are we able to skip the null check on the PhotoUpload property?
  - Briefly explain the steps necessary to deliver both the Vehicle Index page and the vehicle photos displayed on that page. Please mention what requests are made from the browser and what the HTML output would look like.