Unit 07 File Uploads

What is a MIME type?

A **media type** (also known as a **Multipurpose Internet Mail Extensions or MIME type**) is a standard that indicates the nature and format of a document, file, or assortment of bytes. It is defined and standardized in IETF's RFC 6838.

The Internet Assigned Numbers Authority (IANA) is responsible for all official MIME types, and you can find the most up-to-date and complete list at their Media Types page.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

Structure of a MIME type

The simplest MIME type consists of a type and a subtype; these are both strings which, when concatenated with a slash (/) between them, comprise a MIME type. No whitespace is allowed in a MIME type:

```
type/subtype
```

The **type** represents the general category into which the data type falls, such as `video` or `text`. The **subtype** identifies the exact kind of data of the specified type the MIME type represents. For example, for the MIME type `text`, the subtype might be `plain` (plain text), `html` (HTML source code), or `calendar` (for iCalendar/`.ics`) files.

Each type has its own set of possible subtypes, and a MIME type always has both a type and a subtype, never just one or the other.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

Common MIME Type Categories

text, image, audio, video, font, application

Common MIME Types

text/plain - plain text file (.txt)
text/html - HTML file (.htm or .html)
text/xml - XML file (.xml)
image/jpeg - JPEG image file (.jpg or .jpeg)
image/png - PNG image file (.png)
audio/mpeg - MP3 audio file (.mp3)
video/mp4 - MP4 video file (.mp4)
application/pdf - PDF file (.pdf)
application/json - JSON file or JSON response (.json)

MIME Type Resources

MDN Mime Types
https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

IANA Official Media Types
https://www.iana.org/assignments/media-types/media-types.xhtml

Microsoft Office MIME Types
https://blogs.msdn.microsoft.com/vsofficedeveloper/2008/05/08/office-2007-file-format-mime-types-for-http-content-streaming-2/

HTML

- ❏ To upload files with an HTML form you must set the **enctype to "multipart/form-data"**

- ❏ An input tag with type **"file"** will allow the user to select one or more files that they want to upload from their computer.

- ❏ Use the **accept** attribute to specify what kinds of files the user can upload.
  - ❏ To specify more than one value, separate the values with a comma.
  - ❏ **image/*** to allow all image files
  - ❏ **audio/*** to allow all audio files
  - ❏ **video/*** to allow all video files
  - ❏ **.gif, .jpg, jpeg, .png** to only allow commonly supported image files.
  - ❏ **.csv, .xls, .xlsx** to only allow common spreadsheet formats.
  - ❏ **.pdf, .txt, .rtf, .doc, .docx** to only allow common text document formats.

- ❏ Use the **multiple** attribute to allow the user to upload multiple files at once.

```
<form action="/Upload" method="post" enctype="multipart/form-data">
  <label for="MyFiles">File to upload</label>
  <input type="file" id="MyFiles" name="MyFiles" accept="image/*" multiple>
  <button type="submit">Upload</button>
</form>
```

HTML Resources

MDN <input type="file">
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/file
HTML <form> enctype Attribute
https://www.w3schools.com/tags/att_form_enctype.asp
HTML <input type="file">
https://www.w3schools.com/tags/att_input_type_file.asp
HTML <input> accept Attribute
https://www.w3schools.com/tags/att_input_accept.asp
Microsoft Office MIME Types
https://blogs.msdn.microsoft.com/vsofficedeveloper/2008/05/08/office-2007-file-format-mime-types-for-http-content-streaming-2/

Styling File Inputs

By default the look of file inputs depends very highly on the user's OS and browser. The problem is that every browser has a different way of implementing <input type="file" /> and only a few can be touched with CSS.

The best technique for styling file uploads today, revolves around exploiting one of the commonly forgotten behaviors of **labels.** Interacting with a **label** triggers click and focus events on the bound input. This works even if the input field is not visible to the user.

The simplest version of this technique is shown below.

HTML

```
<input type="file" id="MyFile" name="MyFile" />
<label for="MyFile" />choose a file</label>
```

CSS

```
[type="file"] {
    width: 0;
    height: 0;
    overflow: hidden;
}
[type="file"] + label {
    cursor: pointer;
    display: inline-block;
    color: #fff;
    background-color: #009900;
    border-radius: 8px;
    padding: 0.5rem 2rem;
}
```

Read the articles below for more in depth techniques and styling tips.

> **Styling File Inputs - The Best Way**
> https://benmarshall.me/styling-file-inputs/
> **Styling & Customizing File Inputs the Smart Way**
> https://tympanus.net/codrops/2015/09/15/styling-customizing-file-inputs-smart-way/

Bootstrap also provides tools to style file inputs, but read the above articles first for tips on making it more user-friendly with JavaScript and to avoid common pitfalls.

```
<div class="input-group mb-3">
  <div class="custom-file">
    <input type="file" class="custom-file-input" id="MyFile" name="MyFile">
    <label class="custom-file-label" for="MyFile">Choose file</label>
  </div>
</div>
```

> **Bootstrap Input Groups**
> https://getbootstrap.com/docs/4.0/components/input-group/
> **Bootstrap Custom File Inputs**
> https://getbootstrap.com/docs/4.0/components/input-group/#custom-file-input

ASP.NET

- ❏ To receive uploaded files on the backend. Your action method must include a parameter with the same name as the input field. That parameter must be of type **HttpPostedFileBase** if you wish to accept a single file from the user.

- ❏ To receive multiple files from the user the parameter should be of type **HttpPostedFileBase[]**

- ❏ Do not use **HttpPostedFile!** This class is a relic of ASP.NET Classic and does not work in MVC. **HttpPostedFileBase** was created to support unit testing.

- ❏ If a page refresh occurs then browser will forget what files the user selected. So when uploaded files it is highly recommended to use **AJAX.**

- ❏ Sending the files to the server over **AJAX** also allows us to listen to upload progress events and provide a progress bar to the user. As we will discuss shortly.

```
[HttpPost]
public async Task<ActionResult> Upload(MyModel model, HttpPostedFileBase[] myFiles)
{
  if (!ModelState.IsValid) {
    return Json(new { Success = false, Message = "Invalid Data"; });
  }

  // save model to database
  ...

  // upload files to server
  foreach (HttpPostedFileBase file in files)
  {
    if (file != null)
    {
        var inputFileName = Path.GetFileName(file.FileName);
        var serverPath = Path.Combine(Server.MapPath("~/UploadedFiles/"), inputFileName);
        file.SaveAs(serverPath);
    }
  }

  // success
  string msg = files.Length + " files uploaded successfully.";
  return Json(new { Success = true, Message = msg; });
}
```

HttpPostedFileBase Class
https://docs.microsoft.com/en-us/dotnet/api/system.web.httppostedfilebase
Uploading Multiple Files In ASP.NET MVC
https://www.c-sharpcorner.com/article/uploading-multiple-files-in-asp-net-mvc/

Working with Paths

When working with files we frequently find the need to work with paths, filenames, and extensions.

> **Virtual Path** - The path to the file from the perspective of the browser. (Usually starts with **"~/"**)
>
> **Physical Path** - The path to the file on the server's hard drive. (Usually starts with **"C:\"**)
>
> **Absolute Path** - The complete path to a file all the way back to the root. For paths to files on disk these typically start with **"C:\"** on Windows or **"/"** on Linux.
>
> **Relative Path** - The partial path to a file relative to the current directory.
>
> **File Name** - The name of a file including the extension.
>
> **Extension** - The sequence of letters that come after the last period in a file name. Extension are optional, most files do have an extension that specifies what kind of file it is, but it is not strictly mandatory. Extensions are usually 1 to 4 characters in length, but may be arbitrarily long.

The **Path** class in **C#** provides a lot of useful methods for working with paths.

> **Path.Combine()**
> Concatenate paths together.
> **Path.GetFileName()**
> Returns the file name that the path refers to.
> **Path.GetExtension()**
> Returns the extension (including the period) of the file specified in the path.

The **Server.MapPath()** method can be used to transform a virtual path into a physical path. (However this method is only available within Controllers and Views.)

Path Class
https://docs.microsoft.com/en-us/dotnet/api/system.io.path
MapPath() method
https://docs.microsoft.com/en-us/dotnet/api/system.web.httpserverutility.mappath

Using jQuery to listen for progress events

- ❏ You can use the XHR object to listen for download progress events.
- ❏ The **progress** event is fired as files are uploaded and provides you with the number of bytes that have been uploaded thus far.
- ❏ The **load** event is fired when all of the files have been uploaded, but before a response is returned.
- ❏ The shorthand methods **$.get()** and **$.post()** do not provide this access to the XHR object. To listen to progress events you must use the long form **$.ajax()** method.

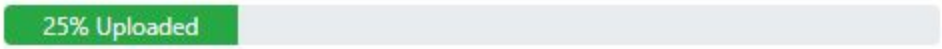
```
$.ajax({
  ...
  xhr: () => {
    const xhr = new window.XMLHttpRequest();
    xhr.upload.addEventListener("progress", evt => {
      if (evt.lengthComputable) {
        const percent = Math.floor(100 * evt.loaded / evt.total);
        console.log(`uploading... {percent.toFixed(0)}%`);
      }
    }, false);
    xhr.upload.addEventListener("load", evt => {
      console.log(`upload complete`);
    }, false);
    return xhr;
  }
})
```

jQuery.ajax()
https://api.jquery.com/jquery.ajax/
MDN Using files from web applications
https://developer.mozilla.org/en-US/docs/Web/API/File/Using_files_from_web_applications
MDN toFixed() method
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toFixed

Bootstrap Progress Bars

https://getbootstrap.com/docs/4.3/components/progress/

Bootstrap provides a browser independent progress bar widget. Example HTML provided below. You will need to dynamically update the value & width of the progress bar through JavaScript as shown below.



Example HTML

```
<div id="MyProgressBar" class="progress" style="height: 20px;">
  <div class="progress-bar bg-success"
       style="width: 0%;"
       role="progressbar"
       aria-valuenow="0"
       aria-valuemin="0"
       aria-valuemax="100">
    0% Uploaded
  </div>
</div>
```

Example JavaScript

```
let percent = 0;
setInterval(() => {
    ++percent;
    const percentString = percent.toFixed(0);
    $('#MyProgressBar .progress-bar')
        .css('width', percentString + '%')
        .attr('aria-valuenow', percentString)
        .html(percentString + '% Uploaded');
}, 100);
```

Scaling Images and Making Thumbnails

The **WebImage** class is part of the ASP.NET framework and provides utilities to validate images, resize them, and save them to disk.

```
private void SaveImage(
    HttpPostedFileBase file,
    string destinationFolder,
    string filename,
    int maxWidth,
    int maxHeight)
{
    // Save the image
    WebImage img = new WebImage(file.InputStream);
    if (img.Width > maxWidth || img.Height > maxHeight)
    {
        img.Resize(maxWidth, maxHeight);
    }
    img.Save(Path.Combine(destinationFolder, filename));

    // Save the thumbnail
    img.Resize(THUMBNAIL_IMAGE_SIZE, THUMBNAIL_IMAGE_SIZE);
    img.Save(Path.Combine(destinationFolder, "Thumbnails/", filename));
}
```

WebImage Class
https://docs.microsoft.com/en-us/dotnet/api/system.web.helpers.webimage

Loading Data from Excel Spreadsheets

ExcelDataReader on nuget
https://www.nuget.org/packages/ExcelDataReader/
ExcelDataReader on github
https://github.com/ExcelDataReader/ExcelDataReader
DataSet Class (part of ADO.NET)
https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset

```
private DataSet LoadFileIntoDataSet(string path, bool hasHeaderRow = true)
{
    Func<Stream, IExcelDataReader> createReader;

    if (Path.GetExtension(path).ToLower() == ".xlsx" ||
        Path.GetExtension(path).ToLower() == ".xls")
    {
        createReader = stream => ExcelReaderFactory.CreateReader(stream);
    }
    else if (Path.GetExtension(path).ToLower() == ".csv")
    {
        createReader = stream => ExcelReaderFactory.CreateCsvReader(stream);
    }
    else
    {
        throw new Exception("File is neither a .csv, .xls, or .xlsx file");
    }

    using (var stream =
           System.IO.File.Open(path, FileMode.Open, FileAccess.Read, FileShare.Read))
    {
        using (var reader = createReader(stream))
        {
            var dataSet = reader.AsDataSet(new ExcelDataSetConfiguration
            {
                ConfigureDataTable = _ => new ExcelDataTableConfiguration
                {
                    UseHeaderRow = hasHeaderRow
                }
            });
            return dataSet;
        }
    }
}
```

Importing Data from Excel Spreadsheets

Rough Outline

Step 1. Install **ExcelDataReader** package from nuget
Step 2. Load data into a **DataSet** as shown above.
Step 3. Create a **[ViewModel]** class to represent each row of the imported spreadsheet.
Step 4. Iterate over the rows of the **DataSet** and transform each row into a **[ViewModel]** object.
Step 5. Use that sequence of **[ViewModel]** objects to populate necessary table(s) in the database.

Working with DataSets

dataSet.Tables.Count the number of tabs in the spreadsheet
dataSet.Tables[0] the first tab in the spreadsheet

table.Columns.Count the number of columns in a table
table.Columns[0] the first column in a table
table.Column[0].ColumnName the name of the first column in a table
table.Rows.Count the number of rows in a table
table.Rows[0] the first row in a table

table.Rows[i][j].ToString() the text contained in the cell on the **i-th** row and the **j-th** column

Example: Importing Basic Customer Information

```
public class ImportedCustomer
{
    public int RowNumber { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
}

private void ImportCustomerData(DataSet dataSet)
{
    // Verify that there is only one tab in the spreadsheet
    if (dataSet.Tables.Count != 1)
    {
        throw new Exception($"Expected one sheet in Excel file, found {dataSet.Tables.Count}");
    }

    // Get the first tab of the spreadsheet
    DataTable table = dataSet.Tables[0];

    // Verify that the columns are what we expect
    if (table.Columns.Count != 3 ||
        table.Columns[0].ColumnName != "FirstName" ||
        table.Columns[1].ColumnName != "LastName" ||
        table.Columns[2].ColumnName != "Email")
```

```
    {
        throw new Exception("Expected columns FirstName, LastName, Email");
    }

    // Load data from the table into memory
    var importedCustomers = new List<ImportedCustomer>();
    int rowNumber = 1;
    foreach (DataRow row in table.Rows)
    {
        ++rowNumber;
        var customer = new ImportedCustomer
        {
            RowNumber = rowNumber,
            FirstName = row[0].ToString().Trim(),
            LastName = row[1].ToString().Trim(),
            Email = row[2].ToString().Trim(),
        };
        importedCustomers.Add(customer);
    }

    // Load existing customer data from the database
    var existingCustomers = (
        from c in db.Customers
        join i in importedCustomers
            on new { c.FirstName, c.LastName, c.Email }
            equals new { i.FirstName, i.LastName, i.Email }
        select c
    ).ToList();

    // Filter out existing customers
    var newCustomers =
        from i in importedCustomers
        join e in existingCustomers
            on new { c.FirstName, c.LastName, c.Email }
            equals new { i.FirstName, i.LastName, i.Email }
            into g
        where !g.Any()
        select i;

    // Save only the new customers
    db.Customers.AddRange(newCustomers);
    db.SaveChanges();
}
```