Take Home Challenge Scenario

Summary

Create an ASP.NET Core 6 MVC Web Application

The application is to provide the following features for its users:

- Allow users to see a list of video files that have been uploaded to a server media folder
- Allow users to upload new MP4 video files into the server media folder
- Allow users to playback any MP4 video file that is in the server media folder

Take Home Challenge Requirements

- The web application does not require authentication or authorization.
- The web application is to provide all its view(s) from one single URL - Home/Index. The default path should direct to Home/Index, so the path can be omitted, and the site can be accessed simply from 'https://localhost:{portNum}/'
- The web applications front-end should provide two discrete actions that are mutually exclusive - Uploading file(s) - from an upload form view and viewing the contents of the catalogue available in the server media folder - from a catalogue view.
- The upload form view should allow the user to browse their clients file system and choose one or more MP4 files. Once files have been selected, they can simply request the files be uploaded to the server's media folder by pressing an upload button.
- The catalogue view should present all the MP4 files in the server's media folder in a table containing the filename and file size of each file in the catalogue.
- When the user clicks on an item in the catalogue view table, the video should commence playing back in a video player on the same page. If the user selects another video from the table, it closes the previous one and starts playing back the latest selected one. Only one video can ever be seen playing back at a time.
- The web application should also host an ASP.NET Core Web API (please do not use minimal APIs for this). This Web API should be used by the upload form for uploading content to the server's media folder.
- The upload API should support uploading one or more MP4 files at a time. If a file with the same name is uploaded more than once, you can overwrite the pre-existing file with the new one.
- Only files with an MP4 extension are allowed to be uploaded into the server's media folder.
- The upload API should accept uploads of up to 200 megabytes, no more. This limit should be enforced for the upload API only. Uploads greater than 200MB should return an appropriate error code. Any other endpoints exposed by the web application should continue to enforce the default ASP.NET upload limit (which is much less than 200MB).
- After the upload form is used to successfully upload content, the user should be shown the updated catalogue view.
- If an error occurs whilst uploading files to the catalogue, the view should stay on the upload form view and inform the user that an error has occurred.
- The request to upload content and access content for playback must work as if it were deployed on a remote web server, with relative references to the media from the origin host of the deployed web application. You can assume though that the permissions to the media folder would just work if deployed on a remote server (even though there would be extra configuration required for a real deployment). To clarify - what this means is that the web browser should access video files when playing back no different to how it requests CSS, JavaScript or image files. They should be delivered by the web server as static resource files.

Style:

- You are free to style the front-end however you wish, feel free to be creative and implement a theme of some sort.
- The web application is to have a responsive front-end, supporting widths ranging from 400px to 1400px. There are to be NO horizontal scroll bars within this range. Height can overflow with vertical scroll bars.
- The video is to be presented on the page in a consistent size, no matter the dimensions of the source video stream.
- You are free to use any front-end libraries you wish. Please note: A 'library' is not a 'framework'. This means that using jQuery, Bootstrap, KnockoutJS, etc. are fine. Using React, Angular, etc. are out of scope. Writing vanilla JS is fine (make sure it is clean and concise though).
- Feel free to author your scripts in Typescript if you like, provided the transpile step only takes place at publish time - no runtime server scripting please (this means no nodejs at runtime).

Build:

- The expectation is that we can open the provided solution in Visual Studio 2022, with the .NET 6 SDK & runtime, and optionally Typescript installed - and just run your web application locally using the likes of Kestral (eg. dotnet run) or IIS Express in Visual Studio. Other than NuGet restore, or a tsc, there shouldn't be any need for anything other than the ASP.NET Core application to compile.

Take Home Challenge Results

- Share all source files so we can compile, build and give the program a go!
- Let us know how much time you spent on this challenge
- The goal of this exercise is to get a feel for how you would go about designing and building a component and then be able to discuss it and listen/respond to any feedback. We will review and ask questions about the design/coding choices and ask questions about potential improvements. This will be more of an open-ended conversation that simulates how we do code reviews as part of our standard development practices