Challenge 7

By: Mosh Hamedani

Uploading Files

For the next week, I want you to add the ability to upload photos for each vehicle. Now, we have two choices for implementing this feature.

One approach is to extend the vehicle form and add a section for uploading photos. While this approach is technically feasible, it brings some additional complexities and edge cases:

- What if the user enters data for a new vehicle, uploads a few photos, but decides to discard the changes? We don't want these photos to remain in a permanent storage on the server. So we should save them in a temporary space, and when the user clicks the save button, we move them to a permanent location.

- What if the user edits an existing vehicle and uploads a new photo but forgets to click the save button? When a user sees a preview of the image they uploaded, they see this as a confirmation that data is saved on the server. So, they may not intuitively assume that they have to make an extra step and click the save button. This can cause some usability issues.

- If clicking the save button is going to be optional, this will be inconsistent with filling out the rest of the form. For other pieces of information about this vehicle (eg. make, model, etc), the user should click the save button, but for the photos, they don't have to. This is confusing to a lot of users.

- If we enforce them to click the save button, that means we need to take extra care when managing photos. If the user deletes one of the existing photos, we shouldn't actually delete the photo in the permanent storage. We should mark it as deleted, and when the save button is clicked, we should actually delete the file.

These are just a few scenarios. The more you think about this approach, the more usability and technical issues you can discover. Sure, we can still go ahead and use this approach, but the end result will be complex code with many edge cases, bugs and potentially a confusing user interface.

Whenever you find yourself in situations like this, take a break and start over. Ask yourself: "Is there a better way to solve this problem?"

Challenge 7

By: Mosh Hamedani

Here is an alternative approach. We keep the vehicle form as is. When the user clicks the save button to add a new vehicle, we redirect them to a new page called View Vehicle page. On this page, we'll have two tabs: Basics and Photos. On the basics page, we'll have two buttons: Edit and Delete. Clicking the Edit button takes the user back to the vehicle form. Any changes the user makes to the vehicle, they should click the save button. So far so good.

Back ti our View Vehicle page, we also have a second tab: Photos. On this tab, the user can upload photos or delete them. There is no save button. Whenever they upload a photo, they see the thumbnail and this confirms that the photo is uploaded. We don't have to manage two photo storage locations (temporary and permanent). Any changes made by the user take effect immediately. See the difference?

So, use the second approach to implement uploading files.

To save the photo on the server, create an API just like before. This endpoint should take two parameters: vehicleId and file data. File data should be of type **IFormFile**. This is a new interface in ASP.NET Core that gives you access to the posted file. Read about this interface and how you can get the file data and save it to the disk.

Currently, we cannot generate thumbnails with ASP.NET Core because **System.Drawing** namespace is not implemented in .NET Core yet. So, ignore this part for now.