CSE686 Midterm #2

Name: _____Instructor's Solution_____ SUID: _____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All Exams will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be the easiest.

1. What is dependency injection? How does it work? If you create an Asp.Net Core service, how will you arrange to have it injected to using code.

   Dependency Injection is a process of creating instances of services bound to an interface they implement. Injection occurs when the framework passes a reference of the service to an application method that has a parameter of the interface type. That method depends on the interface contract, not the service implementation.

   Asp.Net Dependency Injection is implemented with an Object Factory used by the pipeline infrastructure to return references of service instances to pipeline processing components, as well as controllers and models. The service interfaces and service types are registered with a Dependency Injection Container in the

   ```
   Startup.ConfigureServices(IServiceCollection services)
   ```

   method, using one of the statements:

   ```
   services.AddSingletonService<IServiceType, ServiceType>();
   services.AddScopedService<IServiceType, ServiceType>();
   services.AddTransientService<IServiceType, ServiceType>();
   ```

   SingletonService has the lifetime of the application. ScopedService has the lifetime of the HTTP request, and TransientService has the lifetime of the Service request.

   Services are injected into the

   ```
   Startup.Configure(
         IApplicationBuilder app, IHostingEnvironment env,
         IServiceType svc, …
   )
   ```

   method, with one service reference parameter for each injected service. In this method, pipeline components are created and configured by the app.Use method and app.UseXXX externsions and have access to the injected services they need.

   To define an application's custom services, create a Services folder under the project, add an Interface file and a service file that implements the interface. To register the custom service you need to include a using statement for the Services folder in the Startup.cs file and register as described above. To use the service in a controller, for example, add a using statement for the Services folder to the controller file, and add a controller constructor that accepts a service instance bound to the interface. The infrastructure will inject a service instance bound to the interface when it constructs the controller.

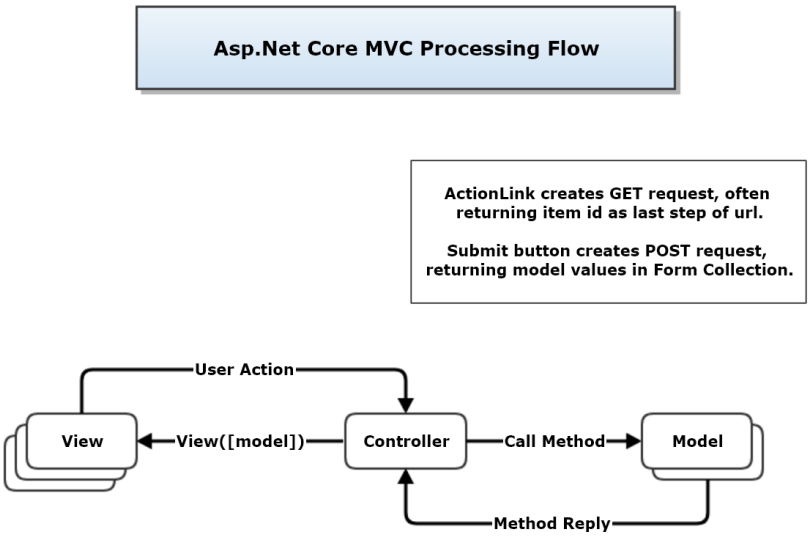
2. Write all the code and markup to display, in a web page, the following:
   - An image with a caption.
   - Two boxes, one red and one green, that are adjacent, below the image.
   - Clicking on the red box decreases the image size by a factor of 1.2.
   - Clicking on the green box increases the image size by a factor of 1.2.

```
  <style>
    body {
      padding: 2em 5em;
      font-family: 'Comic Sans MS';
      font-size: large;
    }
    .ButtonContainer {
      margin-bottom: 1em;
      font-weight: bolder;
    }
    .GreenButton {
      padding: 0.5em 1.5em;
      background-color: green;
      user-select: none;
      -moz-user-select: none;
      -webkit-user-select: none;
      -ms-user-select: none;
    }
    .RedButton {
      padding: 0.5em 1.5em;
      background-color: red;
      user-select:none;
      -moz-user-select:none;
      -webkit-user-select:none;
      -ms-user-select:none;
    }
    .Spacer {
      padding: 0.5em;
    }
  </style>
  <script>
    function bigger() {
      var elem = document.getElementById("theImg");
      elem.width *= 1.2;
    }
    function smaller() {
      var elem = document.getElementById("theImg");
      elem.width *= 1 / 1.2;
    }
  </script>
</head>
<body>
  <div class="ButtonContainer">
    <span class="GreenButton" onclick="bigger()">Bigger</span>
    <span class="Spacer"></span>
    <span class="RedButton" onclick="smaller()">Smaller</span>
  </div>
  <div>
    <img id="theImg" src="CapeCod6.jpg" width="600"/>
  </div>
  <div>
    Cape Cod Beach, North of Eastham, MA.
  </div>
</body>
```

Bigger Smaller

Cape Cod Beach, North of Eastham, MA.

3. Describe the sequence of all activities that occur when a user clicks an ActionLink in a View. Please consider all the cases you can think of (there are not many).



When a user clicks on an ActionLink a get request is sent to the controller action specified in the link, e.g.:

```
@Html.ActionLink("Details", "Details", new { id = item.Id })
```

This link sends a get request to the Details action of the controller. It is supplying an id to identify the element for which details are to be shown in a returned Details view.

The action will usually get data for a new view from a model and return the new view, with the model data, to the browser, e.g. **`return View(model)`**.

Here are the cases:

- No id is sent, because the result should be a view like Index that displays a collection
- An id is sent, because the result concerns one item of a collection in the current view
- A controller specification is declared which causes the action to be processed in a different controller.
- A redirection occurs, perhaps due to an error processing the get request.