```
@inherits SpecFlow.Plus.Runner.Reporting.CustomTemplateBase<TestRunResult>

@using System
@using System.Collections.Generic
@using System.Linq
@using System.Globalization
@using TechTalk.SpecRun.Framework
@using TechTalk.SpecRun.Framework.Results
@using TechTalk.SpecRun.Framework.TestSuiteStructure
@helper GetReportBar(TestItemExecutionResult test)
    {
    <td class="testview-item"
        data-sr-exectime="@GetSeconds(test.ExecutionTime)"
        data-sr-rescode="@((int)test.ResultType)" data-sr-order="@test.ExecutionOrder"
        data-sr-acttime="@GetSeconds(test.ActExecutionTime)">
        <a
            class="bar @("color" + test.ResultType)"
            style="height: @GetBarSize(tr => tr.ExecutionTime.DurationMilliseconds,
            test, 0.0, 60)px;"
            title="@GetTestBarTooltip(test)"
            href="#@GetTestAnchor(test)"> </a>
    </td>
}
@helper GetTimelineBar(DateTime startTime, DateTime endTime, double msecPerPixel,
TestItemExecutionResult test)
    {
    int endPixel = Math.Max((int)Math.Round((endTime -
    Model.ExecutionTime.StartTime).TotalMilliseconds / msecPerPixel), currentPixel + 4);
    int size = endPixel - currentPixel;
    currentPixel = endPixel;
    <td>
        <a
            class="bar @(test == null ? "startupBar" : "color" +
            test.ResultType.ToString())"
            style="width: @(size - 1)px;"
            @if (test != null)
            {
                @:title="@GetTestBarTooltip(test)"
                @:href="#@GetTestAnchor(test)"
            }
            else
            {
                @:title="test thread startup"
            }
        > </a>
    </td>
}
@helper GetSummaryHeader(string titleHeader, bool showDuration = false)
{
    <tr>
        @if (titleHeader != null)
        {
            <th>@titleHeader</th>
        }
        <th colspan="2">Success rate</th>
        <th>Tests</th>
        <th>Succeeded</th>
        <th>Failed</th>
        <th>Pending</th>
        @if (showDuration)
        {
            <th>Duration</th>
        }
    </tr>
}
@helper GetSummaryRow(TestCollectionResultSummary summary, string title, string href,
TimeSpan? executionDuration = null)
{
    <tr>
```

```
63          @if (title != null)
64          {
65              <td><a href="#@href">@title</a></td>
66          }
67          @RenderTestExecutionSummaryRowTail(summary, executionDuration)
68      </tr>
69  }
70
71  @helper RenderBar(TestNodeResultType testNodeResultType, int count, int total,
    Func<TestNodeResultType, string> titleFactory)
72  {
73      <td>
74          <a class="bar @("color" + testNodeResultType)"
75             style="width: @GetPixelBarWidth(total, count)px;"
76             title="@titleFactory(testNodeResultType)"
77             @if (testNodeResultType.GetGroup() == TestNodeResultTypeGroup.Failure)
78             {
79                 @:href="#error_summary"
80             }
81          ></a>
82      </td>
83  }
84
85  @helper RenderSummaryBars(IEnumerable<KeyValuePair<TestNodeResultType, int>> data, int
    total, Func<TestNodeResultType, string> titleFactory)
86  {
87      <table class="timelineview" cellpadding="0" cellspacing="0">
88          <tr>
89              @foreach (var resultCount in data)
90              {
91                  @RenderBar(resultCount.Key, resultCount.Value, total, titleFactory)
92              }
93          </tr>
94      </table>
95  }
96
97  @helper RenderTotalSummaryRowTail(TestCollectionResultSummary summary, TimeSpan?
    executionDuration = null)
98  {
99      @RenderSummaryRowTail(summary.Total, summary.ResultCounts, summary.TotalMessage,
        summary.GetText, executionDuration)
100 }
101
102 @helper RenderTestExecutionSummaryRowTail(TestCollectionResultSummary summary, TimeSpan?
    executionDuration = null)
103 {
104     @RenderSummaryRowTail(summary.Total, summary.TestExecutionResultCounts,
        summary.TotalMessage, summary.GetText, executionDuration)
105 }
106
107 @helper RenderSummaryRowTail(int total, IDictionary<TestNodeResultType, int>
    resultCounts, string totalMessage, Func<TestNodeResultType, string> getText, TimeSpan?
    executionDuration = null)
108 {
109     int succeeded = resultCounts.Where(rc =>
        rc.Key.IsInGroup(TestNodeResultTypeGroup.Success)).Sum(rc => rc.Value);
110     int failed = resultCounts.Where(rc =>
        rc.Key.IsInGroup(TestNodeResultTypeGroup.Failure)).Sum(rc => rc.Value);
111     int pending = resultCounts.Where(rc =>
        rc.Key.IsInGroup(TestNodeResultTypeGroup.Pending)).Sum(rc => rc.Value);
112
113     <td class="percentage">
114         @if (succeeded + failed + pending > 0)
115         {
116             @:@GetRoundedSuccessPercentage(succeeded, failed, pending)%
117         }
118         else
119         {
120             @:n/a
```

```
121             }
122         </td>
123         <td class="bartd">
124             @RenderSummaryBars(GetOrderedBarChartData(resultCounts), total, getText)
125         </td>
126         <td>@totalMessage</td>
127         <td>@succeeded</td>
128         <td>@failed</td>
129         <td>@pending</td>
130         if (executionDuration != null)
131         {
132             <td>@executionDuration.Value</td>
133         }
134     }
135
136     @helper TestItemLinks(TestItem testItem, int level)
137     {
138         if (level == 0)
139         {
140             <a href="#@GetTestNodeAnchor(testItem, "t", 0)">@testItem.Type:
                @testItem.Title</a>
141         }
142         else
143         {
144             <a href="#@GetTestNodeAnchor(testItem, "t", 0)">@testItem.Title</a>
145         }
146         var tiResult = GetTestItemResult(testItem);
147         if (tiResult != null)
148         {
149             foreach (var retry in tiResult.Executions.Skip(1))
150             {
151                 <a href="#@GetTestNodeAnchor(testItem, "t",
                    retry.TestItemExecutionIndex)">retry #@retry.TestItemExecutionIndex</a>
152             }
153         }
154     }
155     @helper TestNodeLinks(TestNode testNode, int level)
156     {
157         if (testNode is TestItem)
158         {
159             @TestItemLinks((TestItem)testNode, level)
160         }
161         if (testNode is TestCollection)
162         {
163             <span>@testNode.Type: @testNode.Title</span>
164             <ul class="subNodeLinks">
165             @foreach (var subTestNode in ((TestCollection)testNode).SubNodes)
166             {
167                 <li>
168                 @TestNodeLinks(subTestNode, level + 1)
169                 </li>
170             }
171             </ul>
172         }
173     }
174     @functions
175     {
176         double GetRoundedSuccessPercentage(int succeeded, int failed, int pending)
177         {
178             double absolute = succeeded + failed + pending;
179             double percent = succeeded / absolute;
180             double scaledRoundedPercent = Math.Round(percent * 100);
181             return scaledRoundedPercent;
182         }
183         string GetFixtureTitle(TestNode fixtureNode)
184         {
185             return fixtureNode.IsDefaultTestTarget ? fixtureNode.Title : string.Format("{0}
                (target: {1})", fixtureNode.Title, fixtureNode.TestTarget);
186         }
```

```
187
188         TimeSpan CalculateDuration(TestNode testNode)
189         {
190             TimeSpan executionDuration = TimeSpan.Zero;
191
192             var testNodeResults = Model.TestExecutionResults.Where(tr =>
                tr.TestItemResult.TestNode == testNode);
193             foreach (var testNodeResult in testNodeResults)
194             {
195                 executionDuration += testNodeResult.ExecutionTime.Duration;
196             }
197
198             return executionDuration;
199         }
200
201         double GetPixelBarWidth(double total, double value)
202         {
203             return Math.Round((value * 200 / total) - 1);
204         }
205
206         IEnumerable<KeyValuePair<TestNodeResultType, int>>
            GetOrderedBarChartData(IEnumerable<KeyValuePair<TestNodeResultType, int>> source)
207         {
208             return source.Where(rc => rc.Value > 0).OrderByDescending(rc =>
                rc.Key.GetGroup() == TestNodeResultTypeGroup.Success ? 1000 : (int)rc.Key);
209         }
210     }
211
212     @section ProjectInformation
213     {
214         <ul>
215             <li>Start Time: @Model.ExecutionTime.StartTime</li>
216             <li>Duration: @Model.ExecutionTime.Duration</li>
217             <li>Test Threads: @Model.TestThreads.Count</li>
218             @if (Model.FrameworkError != null)
219             {
220                 <li><div class="errorMessage">Execution framework error:
                    @(Model.FrameworkError.ToString())</div></li>
221             }
222         </ul>
223     }
224
225     @section TestResultView
226     {
227         <h2>Test Result View</h2>
228         <div id="testview" class="viewbox">
229             <div id="bar-control">
230                 <div id="bar-control-sort">
231                     <label>sort by:</label>
232                     <span class="option"><input type="radio" name="barSortOrder"
                        value="exectime" />Time</span>
233                     <span class="option"><input type="radio" name="barSortOrder"
                        value="acttime" />Act Time</span>
234                     <span class="option"><input type="radio" name="barSortOrder"
                        value="order" />Execution</span>
235                     <span class="option"><input type="radio" name="barSortOrder"
                        value="rescode" checked="checked" />Result</span>
236                     <span class="option"><input type="checkbox" name="barSortDesc"
                        id="barSortDesc" checked="checked" />desc</span>
237                 </div>
238                 <div id="bar-control-heights">
239                     <label>heights:</label>
240                     <span class="option"><input type="radio" checked="checked"
                        name="barHeight" value="exectime" />Time</span>
241                     <span class="option"><input type="radio" name="barHeight"
                        value="acttime" />Act Time</span>
242                 </div>
243             </div>
244             <table class="vertical-scale" cellpadding="0" cellspacing="0">
```

```
245             <tr class="scale-max">
246                 <td class="left-padding scale-max-label"> </td>
247                 <td colspan="@Model.TestExecutionResults.Count()"> </td>
248                 <td class="right-padding"> </td>
249             </tr>
250             <tr class="scale-mid">
251                 <td class="left-padding scale-mid-label"> </td>
252                 <td colspan="@Model.TestExecutionResults.Count()"> </td>
253                 <td class="right-padding"> </td>
254             </tr>
255             <tr class="scale-min">
256                 <td class="left-padding scale-min-label"> </td>
257                 <td colspan="@Model.TestExecutionResults.Count()"> </td>
258                 <td class="right-padding"> </td>
259             </tr>
260         </table>
261         <div class="scrollable">
262             <table class="testview-items" cellpadding="0" cellspacing="0">
263                 <tr class="testview-items-row">
264                     <td class="left-padding"> </td>
265                     @foreach (var test in Model.TestExecutionResults.OrderBy(tr =>
                        tr.ResultType))
266                     {
267                         @GetReportBar(test);
268                     }
269                     <td class="right-padding"> </td>
270                 </tr>
271                 <tr class="horizontal-scale">
272                     <td class="left-padding"> </td>
273                     <td colspan="10"> </td>
274                     @for (int test10Index = 1; test10Index <
                        Model.TestExecutionResults.Count() / 10; test10Index++)
275                     {
276                         <td class="scale-10-label" colspan="10">@(test10Index * 10)</td>
277                     }
278                 </tr>
279             </table>
280         </div>
281     </div>
282 }
283 @section FeatureSummary
284 {
285     <h2>Feature Summary</h2>
286     <table class="testEvents">
287         @GetSummaryHeader("Feature", true)
288 
289         @foreach (var fixtureNode in GetTextFixtures())
290         {
291             var fixtureSummary = GetSummary(fixtureNode);
292             string fixtureTitle = GetFixtureTitle(fixtureNode);
293             string testNodeAnchor = GetTestNodeAnchor(fixtureNode, "f");
294 
295             var executionDuration = fixtureNode.SubNodes.Aggregate(TimeSpan.Zero, (acc,
                testNode) => acc + CalculateDuration(testNode));
296 
297             <tr>
298                 <td><a href="#@testNodeAnchor">@fixtureTitle</a></td>
299                 @RenderTestExecutionSummaryRowTail(fixtureSummary, executionDuration)
300             </tr>
301         }
302     </table>
303 }
304 <!DOCTYPE html>
305 <html>
306     <head>
307         <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
308         <title>@Model.Configuration.ProjectName Test Execution Report</title>
309         <script type="text/javascript"
            src="http://code.jquery.com/jquery-1.6.2.min.js"></script>
```

```
<script type="text/javascript">
    /**
    * jQuery.fn.sortElements
    * --------------
    * #author James Padolsey (http://james.padolsey.com)
    * #version 0.11
    * #updated 18-MAR-2010
    * #url
    https://raw.github.com/jamespadolsey/jQuery-Plugins/master/sortElements/jquer
    y.sortElements.js
    * --------------
    * #param Function comparator:
    *   Exactly the same behaviour as [1,2,3].sort(comparator)
    *
    * #param Function getSortable
    *   A function that should return the element that is
    *   to be sorted. The comparator will run on the
    *   current collection, but you may want the actual
    *   resulting sort to occur on a parent or another
    *   associated element.
    *
    *   E.g. $('td').sortElements(comparator, function(){
    *      return this.parentNode;
    *   })
    *
    *   The <td>'s parent (<tr>) will be sorted instead
    *   of the <td> itself.
    */
    jQuery.fn.sortElements = (function () {

        var sort = [].sort;

        return function (comparator, getSortable) {

            getSortable = getSortable || function () { return this; };

            var placements = this.map(function () {

                var sortElement = getSortable.call(this),
            parentNode = sortElement.parentNode,

                // Since the element itself will change position, we have
                // to have some way of storing it's original position in
                // the DOM. The easiest way is to have a 'flag' node:
            nextSibling = parentNode.insertBefore(
                document.createTextNode(''),
                sortElement.nextSibling
            );

                return function () {

                    if (parentNode === this) {
                        throw new Error(
                    "You can't sort elements if any one is a descendant of another."
                );
                    }

                    // Insert before flag:
                    parentNode.insertBefore(this, nextSibling);
                    // Remove flag:
                    parentNode.removeChild(nextSibling);

                };

            });

            return sort.call(this, comparator).each(function (i) {
                placements[i].call(getSortable.call(this));
            });
```

```
377
378                 };
379
380             })();
381         </script>
382         <script type="text/javascript">
383             jQuery.fn.setBarSizes = (function () {
384                 return function (metricName, maxBarSize, min) {
385                     var max = Math.max.apply(Math, $.makeArray($(this).map(function () {
                        return Number($(this).attr('data-sr-' + metricName)); })));
386                     var scale = 1.0;
387                     while (max > 0.0 && max <= 10.0) {
388                         scale = scale * 10.0;
389                         max = max * 10.0;
390                     }
391                     max = (Math.ceil(max / 2) * 2) / scale;
392
393                     this.each(function () {
394                         var barAnchor = $(this).find('a');
395                         var actual = $(this).attr('data-sr-' + metricName);
396                         var newHeight = Math.max(Math.round(maxBarSize * (actual - min)
                            / (max - min)), 2);
397                         barAnchor.css({ height: newHeight });
398                     });
399
400                     var unit = "";
401                     if (metricName.substr(metricName.length - 4, 4) === "time")
402                         unit = "s";
403
404                     $('#testview .scale-min-label').each(function () {
405                         $(this).text(min.toString() + unit);
406                     });
407                     $('#testview .scale-max-label').each(function () {
408                         $(this).text(max.toString() + unit);
409                     });
410                     var mid = max / 2;
411                     $('#testview .scale-mid-label').each(function () {
412                         $(this).text(mid.toString() + unit);
413                     });
414                 };
415             })();
416
417             function getComparer(metricName, isDesc) {
418                 return function (a, b) {
419                     var aNumber = Number($(a).attr('data-sr-' + metricName));
420                     var bNumber = Number($(b).attr('data-sr-' + metricName));
421                     var result = aNumber > bNumber ? 1 : (aNumber < bNumber ? -1 : 0);
422                     if (isDesc)
423                         result = -1 * result;
424
425                     if (result == 0 && metricName != "order")
426                         result = getComparer("order", false)(a, b);
427
428                     return result;
429                 };
430             }
431
432             var currentSort = "";
433             function doSort(allowToggleDesc) {
434                 var newSort = $("input[name='barSortOrder']:checked").val();
435                 if (allowToggleDesc && currentSort == newSort) {
436                     $('#barSortDesc').click();
437                     doSort(false);
438                     return;
439                 }
440                 currentSort = newSort;
441                 $('#testview td.testview-item').sortElements(getComparer(newSort,
                    $('#barSortDesc').is(':checked')));
442             }
```

```
443
444
445             function doSetHeights(allowSort) {
446                 var selectedMetric = $("input[name='barHeight']:checked").val();
447                 $('#testview td.testview-item').setBarSizes(selectedMetric, 60, 0.0);
448
449                 if (allowSort && currentSort != selectedMetric) {
450                     var $radios = $("input[name='barSortOrder']");
451                     $radios.filter('[value=' + selectedMetric + ']').attr('checked',
                        true);
452                     $("#barSortDesc").attr("checked", [true]);
453                     doSort(false);
454                 }
455             }
456
457             $(document).ready(function () {
458                 $("input[name='barSortOrder']").click(function () { doSort(true); return
                    true; });
459                 $("input[name='barSortDesc']").change(function () { doSort(false); });
460                 $("input[name='barHeight']").change(function () { doSetHeights(true); });
461
462                 doSort(false);
463                 doSetHeights(false);
464
465                 $("div.scrollable").css({ 'overflow': 'auto' });
466             });
467         </script>
468
469         <style type="text/css">
470             body
471             {
472                 color: #000000;
473                 font-family: Arial,Liberation Sans,DejaVu Sans,sans-serif;
474                 line-height: 130%;
475             }
476             h1 {
477                 font-family: Trebuchet MS,Liberation Sans,DejaVu Sans,sans-serif;
478                 font-size: 170%;
479                 font-weight: bold;
480             }
481             h2 {
482                 font-family: Trebuchet MS,Liberation Sans,DejaVu Sans,sans-serif;
483                 font-size: 130%;
484                 font-weight: bold;
485                 margin-bottom: 5px;
486             }
487             h3 {
488                 font-family: Trebuchet MS,Liberation Sans,DejaVu Sans,sans-serif;
489                 font-size: 120%;
490                 font-weight: bold;
491                 margin-bottom: 5px;
492             }
493             a.bar
494             {
495                 text-decoration: none;
496                 display: block;
497                 line-height: 1px;
498             }
499             .description
500             {
501                 font-style: italic;
502             }
503             .log
504             {
505                 width: 600px;
506                 white-space: pre-wrap;
507                 display: block;
508                 margin: 0px;
509             }
```

```
            .errorMessage
            {
                width: 600px;
                color: Red;
                font-weight: bold;
            }
            .stackTrace
            {
                width: 600px;
                white-space: pre-wrap;
                font-style: italic;
                color: Red;
                display: block;
            }
            table.testEvents
            {
                border: solid 1px #e8eef4;
                border-collapse: collapse;
            }
            table.testEvents td
            {
                vertical-align: top;
                padding: 5px;
                border: solid 1px #e8eef4;
            }
            table.testEvents th
            {
                padding: 6px 5px;
                text-align: left;
                background-color: #e8eef4;
                border: solid 1px #a9bfd6;
            }
            .comment
            {
                font-style: italic;
                font-size: smaller;
            }
            .startupBar
            {
                background-color: #EEEEEE;
                cursor: default;
            }

            .colorSucceeded
            {
                background-color: #90ED7B;
            }
            .colorPending
            {
                background-color: #D47BED;
            }
            .colorNothingToRun
            {
                background-color: #CCCCFF;
            }
            .colorInconclusive
            {
                background-color: #7BEDED;
            }
            .colorCleanupFailed
            {
                background-color: #FFCCCC;
            }
            .colorRandomlyFailed
            {
                background-color: #EDB07B;
            }
            .colorFailed
            {
```

```
                background-color: #ED5F5F;
            }
            .colorInitializationFailed
            {
                background-color: #FF0000;
            }
            .colorFrameworkError
            {
                background-color: #FF0000;
            }
            ul.subNodeLinks
            {
                padding-left: 20px;
                margin: 0px;
            }
            ul.subNodeLinks li
            {
                list-style: none;
            }

            /* views general */
            div.scrollable
            {
                /*overflow: auto; - thsi has to be set from js, because of an IE9 bug */
            }
            div.viewbox
            {
                position: relative;
                border: 3px solid #e8eef4;
            }
            div.viewbox table
            {
                border: 0px;
            }

            /* testview */
            #testview
            {
                padding-top: 23px;
            }

            table.testview-items td
            {
                vertical-align: bottom;
                padding: 0px 1px 0px 1px;
            }
            td.right-padding, td.left-padding
            {
                width: 25px;
                min-width: 25px;
            }
            table.testview-items a.bar
            {
                width: 5px;
            }
            table.testview-items tr.testview-items-row
            {
                height: 60px;
            }

            /* scale */
            table.vertical-scale
            {
                position: absolute;
                top: 23px;
                left: 0px;
                width: 100%;
                z-index: -100;
            }
```

```
table.vertical-scale td, tr.horizontal-scale td
{
    font-size: 60%;
    line-height: normal;
}
table.vertical-scale tr.scale-max, table.vertical-scale tr.scale-mid
{
    height: 30px;
}
tr.horizontal-scale, table.vertical-scale tr.scale-min
{
    height: 12px;
}

td.scale-max-label, td.scale-mid-label, td.scale-min-label
{
    border-top: solid 1px #E6E6E6;
    text-align: left;
    vertical-align: top;
}
td.scale-10-label
{
    border-left: solid 1px #E6E6E6;
    text-align: left;
    vertical-align: bottom;
    padding-left: 1px;
}
tr.scale-mid td, tr.scale-min td, tr.scale-max td
{
    border-top: solid 1px #E6E6E6;
}


/* bar-control */
#bar-control
{
    font-size: 60%;
    line-height: normal;
    position: absolute;
    right: 0px;
    top: 0px;
}
#bar-control label
{
    font-weight: bold;
    vertical-align: middle;
}
#bar-control .option
{
    vertical-align: middle;
    text-transform: lowercase;
}
#bar-control input[type="checkbox"]
{
    padding: 0 2px 0 3px;
}
#bar-control input
{
    vertical-align: top;
    height: 12px;
    margin: 0px;
    padding: 0px;
}
#bar-control div
{
    float: right;
    margin: 3px 5px 3px 5px;
}

```

```
            /* timeline view */
            #timelineview
            {
                padding-top: 5px;
            }
            table.timelineview a
            {
                height: 20px;
            }
            table.timelineview td
            {
                vertical-align: bottom;
                padding: 0px 1px 0px 0px;
                border: 0px;
            }
            tr.thread-items-row
            {
                height: 25px;
            }
            tr.thread-items-row td
            {
                vertical-align: bottom;
            }
            td.thread-label
            {
                padding: 0px 6px 0px 6px;
                text-align: right;
                line-height: 18px;
                vertical-align: bottom;
            }
            th.thread-label
            {
                padding: 3px 6px 0px 6px;
                line-height: 18px;
                text-align: left;
                vertical-align: bottom;
            }
            .bartd {
                border-left: 0px solid white !important;
            }
            .percentage {
                border-right: 0px solid white !important;
            }
        </style>
    </head>
    <body>
        <h1>@Model.Configuration.ProjectName Test Execution Report</h1>
        @RenderSection("ProjectInformation")

        <h2>Result: @Model.Summary.ConcludedResultMessage</h2>
        <table class="testEvents">
            @GetSummaryHeader(null)
            @RenderTotalSummaryRowTail(Model.Summary)
        </table>

        <h2>Test Timeline Summary</h2>
        @{
            double msecPerPixel = Model.ExecutionTime.DurationMilliseconds /
            (Model.TestExecutionResults.Count() * 7);
            var secScale = Math.Max(1.0, Math.Round((msecPerPixel / 1000 * 70) / 2) * 2);
            var scaleItemCount = (int)Math.Floor(Model.ExecutionTime.DurationSeconds /
            secScale) + 1;
            var pixelScale = secScale * 1000 / msecPerPixel;
        }
        <div id="timelineview" class="viewbox">
            <div class="scrollable">
                <table cellpadding="0" cellspacing="0">
                    <tr>
                        <th class="thread-label" colspan="2">thread</th>
```

```
                    </tr>
                    @foreach (var testThread in Model.TestThreads)
                    {
                        <tr class="thread-items-row">
                            <td class="thread-label" title="Machine:
                            @testThread.MachineName">#@testThread.ThreadId</td>
                            <td colspan="@scaleItemCount">
                                <table class="timelineview" cellpadding="0"
                                cellspacing="0">
                                    <tr>
                                        @{ currentPixel = 0; }
                                        @GetTimelineBar(Model.ExecutionTime.StartTime,
                                        testThread.ExecutionTime.StartTime,
                                        msecPerPixel, null)
                                        @foreach (var test in
                                        Model.TestExecutionResults.Where(tr =>
                                        tr.ThreadId == testThread.ThreadId).OrderBy(tr
                                        => tr.ExecutionOrder))
                                        {

                                            @GetTimelineBar(test.ExecutionTime.StartTime,
                                             test.ExecutionTime.EndTime, msecPerPixel,
                                            test)
                                        }
                                    </tr>
                                </table>
                            </td>
                        </tr>
                    }
                    <tr class="horizontal-scale">
                        <td> </td>
                        @for (int scaleIndex = 0; scaleIndex < scaleItemCount - 1;
                        scaleIndex++)
                        {
                            var width = (int)(Math.Round((scaleIndex + 1) * pixelScale)
                            - Math.Round((scaleIndex) * pixelScale));
                            <td class="scale-10-label" style="width:
                            @(width)px;min-width: @(width)px;">@Math.Round(secScale *
                            scaleIndex)s</td>
                        }
                        <td class="scale-10-label">@Math.Round(secScale *
                        (scaleItemCount - 1))s</td>
                    </tr>
                </table>
            </div>
        </div>

        @RenderSection("TestResultView")

        @RenderSection("FeatureSummary")

        <a name="error_summary" />
        <h2>Error Summary</h2>
        <table class="testEvents">
            @GetSummaryHeader("Test")
            @foreach (var testResult in Model.Tests.Where(tr => tr.Result.GetGroup() ==
            TestNodeResultTypeGroup.Failure))
            {
                var testSummary = GetSummary(testResult.TestNode);
                <tr>
                    <td>
                        <a href="#@GetTestNodeAnchor(testResult.TestNode, "t",
                        0)">@GetTestTitle(testResult)</a>
                        @foreach (var retry in testResult.Executions.Skip(1))
                        {
                            <a href="#@GetTestAnchor(retry)">retry
                            #@retry.TestItemExecutionIndex</a>
                        }
                    </td>
```

```
                    @RenderTestExecutionSummaryRowTail(testSummary)
                </tr>
                if (!string.IsNullOrEmpty(testResult.Error))
                {
                <tr>
                    <td style="padding-left: 20px;"><div class="errorMessage">Error:
                    @(testResult.Error)</div></td>
                    <td colspan="6">
                        @foreach (string tag in testResult.TestNode.Tags)
                        {
                            if (tag.Contains("Bug") || tag.Contains("bug") ||
                            tag.Contains("BUG") || tag.Contains("FileImport"))
                            {
                                @(tag);
                                <br />
                            }
                        }
                    </td>
                </tr>
                }
            }
        </table>

        <h2>Scenario Summary</h2>
        @foreach (var fixtureNode in GetTextFixtures())
        {
            <a name="@GetTestNodeAnchor(fixtureNode, "f")" />
            <h3>@fixtureNode.Type: @GetFixtureTitle(fixtureNode)</h3>
            if (!string.IsNullOrEmpty(fixtureNode.Description))
            {
                <div class="description"><pre>@fixtureNode.Description</pre>
                </div>
            }
            <table class="testEvents">
                @GetSummaryHeader("Test", true)
                @foreach (var testNode in fixtureNode.SubNodes)
                {
                    var testSummary = GetSummary(testNode);
                    <tr>
                        <td>
                            @TestNodeLinks(testNode, 0)
                        </td>

                        @RenderTestExecutionSummaryRowTail(testSummary,
                        CalculateDuration(testNode))

                    </tr>
                }
            </table>
        }

        <h2>Execution Details</h2>
        @foreach (var test in Model.TestExecutionResults.OrderBy(tr =>
        tr.ExecutionOrder))
        {
            var testItem = test.TestItemResult.TestNode;
            <a name="@GetTestAnchor(test)" />
            <h3>@testItem.Type: @GetTestTitle(test)</h3>
            if (!string.IsNullOrEmpty(testItem.Description))
            {
                <div class="description">
                    <pre>@testItem.Description</pre>
                </div>
            }
            if (testItem.Tags.Any())
            {
                <div class="description">
                    tags: @string.Join(", ", testItem.Tags)
                </div>
```

```
900             }
901             <ul>
902                 <li>Status: @test.ResultType</li>
903                 <li>Start time: @test.ExecutionTime.StartTime</li>
904                 <li>Execution time (sec): @test.ExecutionTime.DurationSeconds</li>
905                 <li>Thread: #@test.ThreadId</li>
906                 @if (!string.IsNullOrEmpty(test.Result.Error))
907                 {
908                 <li>Error: @(test.Result.Error)</li>
909                 }
910             </ul>
911
912             <table class="testEvents">
913                 <tr>
914                     <th>Steps</th>
915                     <th>Trace</th>
916                     <th>Result</th>
917                 </tr>
918                 @foreach (var traceEvent in test.Result.TraceEvents)
919                 {
920                     if (!IsRelevant(traceEvent))
921                     {
922                         continue;
923                     }
924                     var relatedNode = GetTestNode(traceEvent);
925                     <tr>
926                         <td>
927                             <pre class="log">@(GetBusinessMessages(traceEvent))</pre>
928                         </td>
929                         <td>
930                             <!-- [@traceEvent.Type: @relatedNode.Type -
                            @relatedNode.Title] -->
931                             <pre
                            class="log">@Raw(FormatTechMessages(traceEvent.TechMessages.T
                            rimEnd()))</pre>
932                             @if (!string.IsNullOrEmpty(traceEvent.Error))
933                             {
934                                 <div
                                class="errorMessage">@Raw(FormatTechMessages(traceEvent.E
                                rror))</div>
935                                 <pre
                                class="stackTrace">@Raw(FormatTechMessages(traceEvent.Sta
                                ckTrace.TrimEnd()))</pre>
936                             }
937                         </td>
938                         <td>@traceEvent.ResultType in
                        @GetSeconds(Math.Round(traceEvent.Duration.TotalSeconds,
                        3))s</td>
939                     </tr>
940                 }
941             </table>
942         }
943     </body>
944 </html>
```