TASKZERO – **STEP 1**

* To start off, set the version number of the demo. Open **TaskZeroSettings.cs** and change the **Version** constant ant to 1.
* Add the following Nuget packages on the **TaskZero.Server** project
  + MementoFX
  + MementoFX.Persistence.MongoDB
  + MementoFX.Messaging.Postie
  + MementoFX.Messaging.Postie.Unity
* Now let's set up the framework at the start of the application. Create a new file **MementoStartup.cs** in the root of the project and copy the following C# code below in it.

using System;

using Memento.Messaging;

using Memento.Messaging.Postie;

using Memento.Messaging.Postie.Unity;

using Memento.Persistence;

using Microsoft.Practices.Unity;

namespace TaskZero.Server

{

    public class MementoStartup

    {

        public static UnityContainer UnityConfig(Type busType, Type eventStoreType)

        {

            var container = new UnityContainer();

            container.RegisterType<ITypeResolver, UnityTypeResolver>(

new InjectionConstructor(container));

            container.RegisterType(typeof(IBus), busType);

            container.RegisterType(typeof(IEventDispatcher), busType);

            container.RegisterType<IRepository, Repository>(

new InjectionConstructor(eventStoreType));

            container.RegisterType(typeof(IEventStore),

                eventStoreType,

                new InjectionConstructor(typeof(IEventDispatcher)));

            return container;

        }

        public static UnityContainer UnityConfig<TBus, TEventStore>()

        {

            var container = new UnityContainer();

            container.RegisterType<ITypeResolver, UnityTypeResolver>(

new InjectionConstructor(container));

            container.RegisterType(typeof(IBus), typeof(TBus));

            container.RegisterType(typeof(IEventDispatcher), typeof(TBus));

            container.RegisterType<IRepository, Repository>(

new InjectionConstructor(typeof(TEventStore)));

            container.RegisterType(typeof(IEventStore), typeof(TEventStore),

                new InjectionConstructor(typeof(IEventDispatcher)));

            return container;

        }

    }

}

* Add the following properties to the **TaskZeroApplication** class in **global.asax.cs**.

public static IBus Bus { get; private set; }

public static IRepository AggregateRepository { get; private set; }

* Add the following code at the end of the **Application\_Start** method in **global.asax.cs**. The code is meant to initialize the MementoFX framework.

// Configure the MementoFX

var container = MementoStartup.UnityConfig<InMemoryBus, MongoDbEventStore>();

// Save global references to the FX core elements

Bus = container.Resolve<IBus>();

AggregateRepository = container.Resolve<IRepository>();

// Add sagas and handlers to the bus

* Import all necessary namespaces (Visual Studio and/or R# should tell you which) and make sure you also have the following **using** directive.

using Microsoft.Practices.Unity;

* Create a folder **Commands** in the **TaskZero.CommandStack** project and create a C# file in it. Name the file **NotifyCommand.cs**. Make sure you import all references. In particular, you need to reference the MementoFX framework.

using Memento;

namespace TaskZero.CommandStack.Commands

{

    public class NotifyCommand : Command

    {

        public NotifyCommand(string connectionId = "")

        {

            SignalrConnectionId = connectionId;

        }

        public string SignalrConnectionId { get; }

    }

}

Create also a file named **AddNewTaskNotifyCommand.cs** in the same project folder.

using System;

namespace TaskZero.CommandStack.Commands

{

    public class AddNewTaskNotifyCommand : NotifyCommand

    {

        public AddNewTaskNotifyCommand(string connectionId)

            : base(connectionId)

        {

        }

        public Guid TaskId { get; set; }

        public string Title { get; set; }

    }  
}

* Create also a file named **AddNewTaskCommand.cs**. In particular, make sure you reference **TaskZero.Shared**.

using System;

using TaskZero.Shared;

namespace TaskZero.CommandStack.Commands

{

    public class AddNewTaskCommand : NotifyCommand

    {

        public AddNewTaskCommand(string title,

            string description,

            DateTime? dueDate,

            Priority priority,

            string connectionId) : base(connectionId)

        {

            Title = title;

            Description = description;

            DueDate = dueDate;

            Priority = priority;

        }

        public string Title { get; set; }

        public string Description { get; set; }

        public DateTime? DueDate { get; set; }

        public Priority Priority { get; set; }

    }

}

* Create a folder **Events** in TaskZero.Shared and add a **TaskCreatedEvent.cs** file. Make sure you reference MementoFX.

using System;

using Memento;

namespace TaskZero.Shared.Events

{

    public class TaskCreatedEvent : DomainEvent

    {

        public TaskCreatedEvent(Guid id, string title, string description,

DateTime? dueDate, Priority priority)

        {

            TaskId = id;

            Title = title;

            Description = description;

            DueDate = dueDate;

            Priority = priority;

        }

        public Guid TaskId { get; set; }

        public string Title { get; set; }

        public string Description { get; set; }

        public DateTime? DueDate { get; set; }

        public Priority Priority { get; set; }

    }

}

* Create a folder **Model** in the TaskZero.CommandStack project and create a **Task.cs** file in it.

using System;

using Memento.Domain;

using TaskZero.Shared;

using TaskZero.Shared.Events;

namespace TaskZero.CommandStack.Model

{

    public class Task : Aggregate, IApplyEvent<TaskCreatedEvent>

    {

        public Task()

        {

            Priority = Priority.Normal;

            Status = Status.ToDo;

            Enabled = true;

            Deleted = false;

        }

        // COMMON PROPERTIES

        public bool Deleted { get; set; }

        public bool Enabled { get; set; }

        // SPECIFIC PROPERTIES

        public Guid TaskId { get; set; }

        public string Title { get; set; }

        public string Description { get; set; }

        public DateTime? DueDate { get; set; }

        public Priority Priority { get; set; }

        public Status Status { get; set; }

        public void ApplyEvent(

            [AggregateId("TaskId")] TaskCreatedEvent theEvent)

        {

            TaskId = theEvent.TaskId;

            Title = theEvent.Title;

            Description = theEvent.Description;

            DueDate = theEvent.DueDate;

            Priority = theEvent.Priority;

        }

        public static class Factory

        {

            public static Task NewTaskFrom(string title, string descrition,

DateTime? dueDate = null, Priority priority = Priority.Normal)

            {

                var task = new Task();

                var created = new TaskCreatedEvent(Guid.NewGuid(), title, descrition,

dueDate, priority);

                task.RaiseEvent(created);

                return task;

            }

        }

    }

}

* In **Views/Dashboard/index.cshtml** change the *add new task...* button as below:

<a role="button" class="btn btn-primary btn-lg" href="@Url.Action("new", "task")">

    add new task ...

</a>

* Open **Application/ApplicationServiceBase.cs** and replace the content with the following:

using Memento.Messaging.Postie;

namespace TaskZero.Server.Application

{

    public class ApplicationServiceBase

    {

        public ApplicationServiceBase(IBus bus)

        {

            Bus = bus;

        }

        public IBus Bus { get; }

    }

}

* Because of this change—an injected bus parameter—add a new constructor to **DashboardService.cs**.

public DashboardService(IBus bus) : base(bus)

{

}

Edit the initialization of the **DashboardService** instance in **dashboardcontroller.cs** file.

private readonly DashboardService \_service = new DashboardService(TaskZeroApplication.Bus);

* Add a new folder **Task** in the Models folder of the main project and create a file **TaskInputModel.cs** in it.

using System;

using TaskZero.Shared;

namespace TaskZero.Server.Models.Task

{

    public class TaskInputModel

    {

        public TaskInputModel()

        {

            DueDate = null;

            Priority = Priority.NotSet;

            Status = Status.ToDo;

        }

        public Guid TaskId { get; set; }

        public string Title { get; set; }

        public string Description { get; set; }

        public DateTime? DueDate { get; set; }

        public Priority Priority { get; set; }

        public Status Status { get; set; }

        public string SignalrConnectionId { get; set; }

    }

}

In the same folder, also create a **TaskViewModel.cs** file.

namespace TaskZero.Server.Models.Task

{

    public class TaskViewModel : ViewModelBase

    {

    }

}

* Add a new service class in **Application** folder: **TaskService.cs**. Make sure you reference the CommandStack project.

using Memento.Messaging.Postie;

using TaskZero.CommandStack.Commands;

using TaskZero.Server.Models.Task;

namespace TaskZero.Server.Application

{

    public class TaskService : ApplicationServiceBase

    {

        public TaskService(IBus bus) : base(bus)

        {

        }

        #region QUERY methods

        public TaskViewModel GetDefaultTask()

        {

            var model = new TaskViewModel();

            return model;

        }

        #endregion

        #region COMMAND methods

        public void QueueAddOrSaveTask(TaskInputModel input)

        {

            var command = new AddNewTaskCommand(

                input.Title,

                input.Description,

                input.DueDate,

                input.Priority,

                input.SignalrConnectionId);

            Bus.Send(command);

        }

        #endregion

    }

}

* Add **TaskController.cs**.

using System;

using System.Web.Mvc;

using TaskZero.Server.Application;

using TaskZero.Server.Models.Task;

using TaskZero.Shared;

namespace TaskZero.Server.Controllers

{

    [Authorize]

    public class TaskController : AppController

    {

        private readonly TaskService \_service = new TaskService(TaskZeroApplication.Bus);

        #region ADD TASK

        [HttpGet]

        public ActionResult New()

        {

            var model = \_service.GetDefaultTask();

            return View(model);

        }

        [HttpPost]

        public ActionResult Save(TaskInputModel input)

        {

            // If it doesn't crash a serious bus has the message

            // in store and will eventually deliver it.

            // To update the UI, you should actually wait for

            // the operation to complete. It's only started here.

            try

            {

                \_service.QueueAddOrSaveTask(input);

            }

            catch (Exception exception)

            {

                return HandleException(exception);

            }

            // Message delivered

            var response = new CommandResponse(true)

                .SetPartial()

                .AddMessage("Delivered");

            return Json(response);

        }

        #endregion

    }

}

* Create a SignalR hub class in **TaskZero.Shared**. Name it **TaskZeroHub.cs**.

using System;

using Microsoft.AspNet.SignalR;

namespace TaskZero.Shared

{

    public class TaskZeroHub : Hub

    {

        private readonly string \_connectionId;

        public TaskZeroHub(string connectionId)

        {

            \_connectionId = connectionId;

        }

        public void NotifyResultOfAddNewTask(Guid taskId, string title)

        {

            var hubContext = GlobalHost.ConnectionManager.GetHubContext<TaskZeroHub>();

            hubContext

                .Clients

                .Client(\_connectionId)

                .notifyResultOfAddNewTask(taskId.ToString(), title);

        }

        public void NotifyResultOfUpdateTask(Guid taskId, string title)

        {

            var hubContext = GlobalHost.ConnectionManager.GetHubContext<TaskZeroHub>();

            hubContext

                .Clients

                .Client(\_connectionId)

                .notifyResultOfUpdateTask(taskId.ToString(), title);

        }

    }

}

* Create a **Task** folder under Views and add a **new.cshtml** file to it.

@model TaskZero.Server.Models.Task.TaskViewModel

@using TaskZero.Server.Resources

@section adhoc\_Scripts\_Top {

    <script src="~/content/scripts/jquery.signalR-2.2.2.min.js"></script>

    <script src="~/signalr/hubs"></script>

    <script>

        $(function() {

            // Reference the auto-generated proxy for the hub.

            var taskZeroHub = $.connection.taskZeroHub;

            // Define client-side endpoints for the taskZeroHub

            taskZeroHub.client.notifyResultOfAddNewTask = function (taskId, title) {

                var msg = "Task [" + title + "] created successfully.";

                Ybq.toast("#task-form-message", msg, true);

            };

            // Start the SignalR client-side listener

            $.connection.hub.start().done(function() {

                $("#signalrConnectionId").val($.connection.hub.id);

            });

        });

    </script>

}

<div class="col-xs-12 col-sm-8 col-md-6 col-xs-offset-0 col-sm-offset-2 col-md-offset-3">

    <h2>

        <a href="@Url.Action("index", "dashboard")"><i class="fa fa-list"></i></a>

        NEW TASK

    </h2>

    <div id="task-form-message" class="alert alert-info" style="display: none;"></div>

    <div class="margin-top-md">

        <form class="form-horizontal" id="task-form"

              role="form" method="post"

              action="@Url.Action("save", "task")">

            <!-- ID -->

            <input type="hidden" name="signalrConnectionId" id="signalrConnectionId" />

            <!-- Title & Priority -->

            <div class="form-group has-feedback" id="task-form-group-title">

                <label class="col-xs-12 col-md-8" for="title">Task</label>

                <label class="col-xs-12 col-md-4" for="priority">Priority</label>

                <div class="col-xs-12 col-md-8">

                    <input type="text" class="form-control"

                           id="title" name="title"

                           required

                           placeholder="Describe what you should be up to"

                           data-click-on-enter="#task-form-submit-button">

                    <i class="fa fa-edit form-control-feedback"></i>

                </div>

                <div class="col-xs-12 col-md-4">

                    <select name="priority" id="priority" class="form-control">

                        <option value="0">Not Set</option>

                        <option value="1">Low</option>

                        <option value="2">Normal</option>

                        <option value="3">High</option>

                        <option value="4">Urgent</option>

                    </select>

                </div>

            </div>

            <!-- Description & Due date -->

            <div class="form-group" id="task-form-group-description">

                <label class="col-xs-12 col-md-8" for="description">Description</label>

                <label class="col-xs-12 col-md-4" for="duedate">Due date</label>

                <div class="col-xs-12 col-md-8">

                    <textarea class="form-control" rows="5"

                              name="description" id="description"></textarea>

                </div>

                <div class="col-xs-12 col-md-4">

                    <input type="text" class="form-control"

                           id="duedate" name="duedate"

                           date

                           placeholder="Due date">

                </div>

            </div>

            <div class="form-group" style="margin-top: 30px">

                <div class="col-xs-offset-2 col-xs-8 col-md-4 col-md-offset-4">

                    <button type="button" id="task-form-submit-button"

                            class="btn btn-primary btn-block">

                        @Strings\_Menu.Submit

                    </button>

                    <span id="sample-form-loader"

                          class="text-danger" style="display: none;">

                        @Strings\_Core.System\_OperationInProgress

                    </span>

                </div>

            </div>

        </form>

    </div>

</div>

<script>

    $("#task-form-submit-button").click(function() {

        if (Ybq.canAcceptValueOf("#task-form",

            "title",

            function (input) { return input.length > 0; },

            "Title is mandatory")) {

            Ybq.postForm("#task-form",

                function(data) {

                    var response = JSON.parse(data);

                    //Ybq.toast("#task-form-message",

                    //    response.Message, response.Success, response.IsPartial);

                });

        } else {

            Ybq.clearFormAfterTimeout("#task-form");

        }

    });

</script>

* You should be able to build the solution now. If it works, then log in and try to add a new task. Place a breakpoint in the body of the **Save** method of the **TaskController** class and check that posted data arrive correctly. No further effects should be visible at this time.

* Create a folder **Sagas** in TaskZero.CommandStack and add a file named **ManageTaskSaga.cs** to it. Make sure you reference the MementoFX library.

using Memento.Messaging.Postie;

using Memento.Persistence;

using TaskZero.CommandStack.Commands;

using TaskZero.CommandStack.Model;

namespace TaskZero.CommandStack.Sagas

{

    public class ManageTaskSaga : Saga,

        IAmStartedBy<AddNewTaskCommand>

    {

        public ManageTaskSaga(IBus bus, IEventStore eventStore, IRepository repository)

            : base(bus, eventStore, repository)

        {

        }

        public void Handle(AddNewTaskCommand message)

        {

            var task = Task.Factory.NewTaskFrom(

                message.Title, message.Description, message.DueDate, message.Priority);

            Repository.Save(task);

            // Notify back

            var notification = new AddNewTaskNotifyCommand(message.SignalrConnectionId)

            {

                TaskId = task.TaskId,

                Title = task.Title

            };

            Bus.Send(notification);

        }

    }

}

* Add a file **NotificationHandler.cs** to the Sagas folder in the CommandStack project. Make sure you also reference SignalR.Core here.

using Memento.Messaging.Postie;

using TaskZero.CommandStack.Commands;

using TaskZero.Shared;

namespace TaskZero.CommandStack.Sagas

{

    public class NotificationHandler :

        IHandleMessages<AddNewTaskNotifyCommand>

    {

        public void Handle(AddNewTaskNotifyCommand message)

        {

            // Notify back

            var hub = new TaskZeroHub(message.SignalrConnectionId);

            hub.NotifyResultOfAddNewTask(message.TaskId, message.Title);

        }

    }

}

* Open **global.asax.cs** and add the following lines to the end of the **Application\_Start** method.

Bus.RegisterHandler<ManageTaskSaga>();

Bus.RegisterHandler<NotificationHandler>();

* Try running the demo and adding a task. If you have Robo3T installed to explore MongoDB content, you should see a **mfxEventStore** database on **localhost:27017** and under it a **TaskCreatedEvent** collection.

* Now let's proceed to create a read model that reflects the state of the system. Let's start adding the Entity Framework 6.x Nuget package to **TaskZero.ReadStack**. Feel free to remove **app.config**. Note that you need to incorporate part of its content in the **web.config** file of the main server project. However, this has been done already in the parent web.config. Therefore it's already there. **Also make sure you reference the Entity Framework package in the main project.**
* Create a **ReadModel** folder in TaskZero.ReadStack project and add **dto.cs** to it.

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace TaskZero.ReadStack.ReadModel

{

    public class Dto

    {

        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]

        [Key]

        public int Id { get; set; }

    }

}

* Add **PendingTask.cs** to the **ReadModel** folder in the TaskZero.ReadStack project. Make sure you also reference the Shared project.

using System;

using TaskZero.Shared;

namespace TaskZero.ReadStack.ReadModel

{

    public class PendingTask : Dto

    {

        public Guid TaskId { get; set; }

        public string Title { get; set; }

        public string Description { get; set; }

        public DateTime? DueDate { get; set; }

        public DateTime? CompletionDate { get; set; }

        public DateTime? StartDate { get; set; }

        public Priority Priority { get; set; }

        public Status Status { get; set; }

    }

}

* Create a **Repositories** folder in the ReadStack project and add **TaskContext.cs** to it.

using System.Data.Entity;

using TaskZero.ReadStack.ReadModel;

namespace TaskZero.ReadStack.Repositories

{

    public class TaskContext : DbContext

    {

        public TaskContext()

            : base("MfxDemoDb")

        {

        }

        public DbSet<PendingTask> PendingTasks { get; set; }

    }

}

* In the same **Repositories** folder also create a **ProjectionManager.cs** file.

using System;

using System.Linq;

using TaskZero.ReadStack.ReadModel;

namespace TaskZero.ReadStack.Repositories

{

    public class ProjectionManager : IDisposable

    {

        private readonly TaskContext \_context = null;

        public ProjectionManager()

        {

            \_context = new TaskContext();

            \_context.Configuration.AutoDetectChangesEnabled = false;

        }

        public IQueryable<PendingTask> PendingTasks => \_context.PendingTasks;

        public void Dispose()

        {

            \_context?.Dispose();

        }

        public PendingTask FindById(Guid id)

        {

            var task = (from t in PendingTasks

where t.TaskId == id

select t).SingleOrDefault();

            return task;

        }

    }

}

* Create a **Denormalizers** folder in ReadStack and add **ManageTaskDenormalizer.cs** to it. Make sure you also reference **MementoFX** and **Memento.Messaging.Postie**.

using Memento.Messaging.Postie;

using TaskZero.ReadStack.ReadModel;

using TaskZero.ReadStack.Repositories;

using TaskZero.Shared;

using TaskZero.Shared.Events;

namespace TaskZero.ReadStack.Denormalizers

{

    public class ManageTaskDenormalizer :

        IHandleMessages<TaskCreatedEvent>

    {

        public void Handle(TaskCreatedEvent message)

        {

            var task = new PendingTask

            {

                TaskId = message.TaskId,

                Title = message.Title,

                Description = message.Description,

                DueDate = message.DueDate,

                Priority = message.Priority,

                Status = Status.ToDo        // Default status for new tasks (by design)

            };

            using (var context = new TaskContext())

            {

                context.PendingTasks.Add(task);

                context.SaveChanges();

            }

        }

    }

}

* In **global.asax.cs** append the following line to Application\_Start. You need to reference ReadModel.

Bus.RegisterHandler<ManageTaskDenormalizer>();

* Open again **TaskIndexViewModel.cs** in **Models/Home** in the main project. Edit as below:

using System.Collections.Generic;

using TaskZero.ReadStack.ReadModel;

namespace TaskZero.Server.Models.Home

{

    public class TaskIndexViewModel : ViewModelBase

    {

        public TaskIndexViewModel()

        {

            Tasks = new List<PendingTask>();

        }

        public IList<PendingTask> Tasks { get; set; }

    }

}

* Open **DashboardService.cs** in **Application** and edit as below.

using System.Linq;

using Memento.Messaging.Postie;

using TaskZero.ReadStack.Repositories;

using TaskZero.Server.Models.Home;

namespace TaskZero.Server.Application

{

    public class DashboardService : ApplicationServiceBase

    {

        private readonly ProjectionManager \_manager = new ProjectionManager();

        public DashboardService(IBus bus) : base(bus)

        {

        }

        public TaskIndexViewModel GetTaskIndexViewModel()

        {

            var model = new TaskIndexViewModel

            {

                Tasks = (from t in \_manager.PendingTasks select t).ToList()

            };

            return model;

        }

    }

}

* In the main server project, create a new folder **Extensions** under **Common**. Add there the following file **PendingTaskExtensions.cs**.

using System;

using TaskZero.ReadStack.ReadModel;

using TaskZero.Shared;

namespace TaskZero.Server.Common.Extensions

{

    public static class PendingTaskExtensions

    {

        public static string ToColor(this PendingTask pendingTask, Priority priority)

        {

            switch (priority)

            {

                case Priority.Urgent:

                    return "#f00";

                case Priority.High:

                    return "#f80";

                case Priority.Normal:

                    return "#0c0";

                case Priority.Low:

                    return "#0f8";

                default:

                    return "transparent";

            }

        }

        public static DateTime DueDateForDisplay(this PendingTask pendingTask)

        {

            return pendingTask.DueDate ?? DateTime.MaxValue;

        }

        public static string EffortForDisplay(this PendingTask pendingTask)

        {

            var effort = "";

            if (pendingTask.Status == Status.Completed)

            {

                if (pendingTask.StartDate.HasValue && pendingTask.CompletionDate.HasValue)

                {

                    var ts = pendingTask.CompletionDate.Value - pendingTask.StartDate.Value;

                    if (ts.Days <= 0)

                        return "Less than a day";

                    effort = String.Format("{0} day(s)", ts.Days);

                }

            }

            return effort;

        }

    }

}

* Open **pv\_TaskDashboard.cshtml** in **Views/Dashboard**. Add the following directive:

@using TaskZero.Server.Common.Extensions

Replace the code in the subsequent **@{ ... }** block with:

@{

    var pending = (from t in Model.Tasks

                   where t.Status != Status.Completed

                   orderby t.DueDateForDisplay()

                   select t).ToList();

    var completed = (from t in Model.Tasks

                     where t.Status == Status.Completed

                     orderby t.DueDate descending

                     select t).ToList();

}

* Open **pv\_PendingTasks.cshtml** in **Views/Dashboard**. Replace the **@model** directive with:

@model IList<TaskZero.ReadStack.ReadModel.PendingTask>

Append the following code

else

{

    <table class="table table-responsive table-hover hand">

        <thead>

        <tr>

            <td style="width: 10px"></td>

            <td>TASK</td>

            <td>STATUS</td>

            <td>DUE DATE</td>

            <td>&nbsp;</td>

        </tr>

        </thead>

        <tbody>

        @foreach (var task in Model)

        {

            <tr>

                <td style="background: @task.ToColor(task.Priority)"></td>

                <td>

                    <strong>@task.Title</strong><br />

                    <small>@task.Description.ToDefault("N/A")</small>

                </td>

                <td>@task.Status</td>

                <td>

                    @Html.Raw(task.DueDate.HasValue

                        ? task.DueDate.Value.ToString("d MMM yyyy")

                        : "<small class='text-muted'>N/A</small>")

                </td>

                <td>

                    @\*<a role="button" class="btn btn-primary"

                               href="@Url.Action("edit", "task", new {id = task.TaskId})">

                                <i class="fa fa-fw fa-edit"></i>

                            </a>\*@

                    <button class="btn btn-danger">

                        <i class="fa fa-fw fa-trash"></i>

                    </button>

                </td>

            </tr>

        }

        </tbody>

    </table>

}

* Open **pv\_CompletedTasks.cshtml** in **Views/Dashboard**. Replace the **@model** directive with:

@model IList<TaskZero.ReadStack.ReadModel.PendingTask>

Append the following code

else

{

    <table class="table table-responsive">

        <thead>

        <tr>

            <td style="width: 10px"></td>

            <td>TASK</td>

            <td>DUE DATE</td>

            <td>STARTED</td>

            <td>COMPLETED</td>

        </tr>

        </thead>

        @foreach (var task in Model)

        {

            <tr>

                <td style="background: @task.ToColor(task.Priority)"></td>

                <td>

                    <strong>@task.Title</strong><br />

                    <small>@task.Description.ToDefault("N/A")</small>

                </td>

                <td>

                    @Html.Raw(task.DueDate.HasValue

                        ? task.DueDate.Value.ToString("d MMM yyyy")

                        : "<small class='text-muted'>N/A</small>")

                </td>

                <td>

                    @Html.Raw(task.StartDate.HasValue

                        ? task.StartDate.Value.ToString("d MMM yyyy")

                        : "<small class='text-muted'>N/A</small>")

                </td>

                <td>

                    @Html.Raw(task.CompletionDate.HasValue

                        ? task.CompletionDate.Value.ToString("d MMM yyyy")

                        : "<small class='text-muted'>N/A</small>")

                    <div class="text-primary">

                        @task.EffortForDisplay()

                    </div>

                </td>

            </tr>

        }

    </table>

}

* Make sure you have **EntityFramework.dll** and **EntityFramework.SqlServer.dll** in the **Bin** folder of the server project.
* If you may already have a copy of the **mfx\_ReadModel** SQL Server database, delete it for a fresh start of the demo.
* Now try running the app and creating a new task.