TASKZERO – **STEP 2**

At this point, after completing Step 1, you should have a solution that allows you to add a new item to the to-do list. The next step is adding the code to edit an existing item.

- As first thing, let's add some JavaScript code to select the to-do item to edit. The file ybq-coreinit.js contains a jQuery command that automatically captures the click on a TR element and jumps to the URL saved in a child **data-gotourl** attribute. Let's just edit the table in the **pv_pendingtasks.cshtml** file. The folder is Views/Dashboard. Edit the code as shown below.

```
@foreach (var task in Model)
{
          <tr data-gotourl="/task/edit/@task.TaskId">
          ...
}
```

- Now let's proceed creating a new controller action to process the request to edit a to-do item. In the **TaskController** class let's add a new method.

```
#region EDIT TASK
[HttpGet]
public ActionResult Edit(string id /* to bypass model binding and possible exceptions on GUID */)
{
    Guid guid;
    var outcome = Guid.TryParse(id, out guid);

    if (!outcome)
        throw new InvalidGuidException("Could not find specified task");

    var model = _service.GetTask(guid);
    return View(model);
}
#endregion
```

Make sure you also add a reference to the namespace that defines the exception class.

```
using TaskZero.Server.Common.Exceptions;
```

- Open **TaskViewModel.cs** in the Models folder and add the following code.

```
using TaskZero.ReadStack.ReadModel;

namespace TaskZero.Server.Models.Task
{
    public class TaskViewModel : ViewModelBase
    {
        public TaskViewModel()
        {
            Task = new PendingTask();
        }

        public PendingTask Task { get; set; }
    }
}
```

- Open the **TaskService.cs** file and add the following declaration

```
private readonly ProjectionManager _manager = new ProjectionManager();
```

- In the same file, add the following code to load the specified item.

```
public TaskViewModel GetTask(Guid id)
{
    var model = new TaskViewModel { Task = _manager.FindById(id) };
    return model;
}
```

- The method **QueueAddOrSaveTask** in the **TaskService.cs** class must be rewritten to also support the edit scenario.

```
#region COMMAND methods
public void QueueAddOrSaveTask(TaskInputModel input)
{
    Command command;
    var isNewTask = (input.TaskId == Guid.Empty);
    if (isNewTask)
    {
        command = new AddNewTaskCommand(
            input.Title,
            input.Description,
            input.DueDate,
            input.Priority,
            input.SignalrConnectionId);
    }
    else
    {
        command = new UpdateTaskCommand(
            input.TaskId,
            input.Title,
            input.Description,
            input.DueDate,
            input.Priority,
            input.Status,
            input.SignalrConnectionId);
    }

    Bus.Send(command);
}
#endregion
```

- Create also a file named **UpdateTaskCommand.cs** in the Commands folder of the CommandStack project.

```
using System;
using TaskZero.Shared;

namespace TaskZero.CommandStack.Commands
{
    public class UpdateTaskCommand : NotifyCommand
    {
        public UpdateTaskCommand(Guid id,
            string title,
            string description,
            DateTime? dueDate,
            Priority priority,
```

```
            Status status,
            string connectionId) : base(connectionId)
        {
            TaskId = id;
            Title = title;
            Description = description;
            DueDate = dueDate;
            Priority = priority;
            Status = status;
        }

        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; }
    }
}
```

- In the **Views/Task** folder create a new **edit.cshtml** file with the following content.

```
@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.notifyResultOfUpdateTask = function (taskId, title) {
                var msg = "Task [" + title + "] updated 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>
}
@{
    var dateForDisplay = Model.Task.DueDate.HasValue
        ? Model.Task.DueDate.Value.ToString("d MMM yyyy")
        : "";
}
<div class="col-xs-12 col-lg-10 col-lg-offset-1">
    <h2>
        <a href="@Url.Action("index", "dashboard")"><i class="fa fa-list"></i></a>
        EDIT TASK <small class="text-muted hidden-xs">@Model.Task.TaskId</small>
    </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="taskid" value="@Model.Task.TaskId" />
            <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-2" for="priority">Priority</label>
                <label class="col-xs-12 col-md-2" for="priority">Status</label>
                <div class="col-xs-12 col-md-8">
                    <input type="text" class="form-control"
                           id="title" name="title"
                           value="@Model.Task.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-2">
                    <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 class="col-xs-12 col-md-2">
                    <select name="status" id="status" class="form-control">
                        <option value="0">Unknown</option>
                        <option value="1">To do</option>
                        <option value="2">In progress</option>
                        <option value="3">Suspended</option>
                        <option value="4">Completed</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">@Model.Task.Description</textare
a>
                </div>
                <div class="col-xs-12 col-md-4">
                    <input type="text" class="form-control"
                           id="duedate" name="duedate"
                           date
                           value="@dateForDisplay"
                           placeholder="Due date">
                    <h4 id="pending-changes" class="margin-top-md bold text-danger"
                        style="display:none">
                        PENDING CHANGES
                    </h4>
                </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 text-center">
                    <button type="button" id="task-form-submit-button"
                            class="btn btn-primary btn-spaced">
                        @Strings_Menu.Submit
                    </button>   
                    <button type="button" id="task-form-delete-button"
                            class="btn btn-danger btn-spaced"
                            onclick="alert('NOT IMPLEMENTED YET')">
                        @Strings_Menu.Delete
                    </button>
                    <span id="task-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; },
            "Must be non empty")) {
            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>
<script>
    $("#priority").val(@((int)Model.Task.Priority));
</script>
<script>
    $("#status").val(@((int)Model.Task.Status));
</script>
```

- At this point, everything is up and running for editing an existing to-do item. The only missing part is handling the command that requires an update. Let's then open the **ManageTaskSaga.cs** file and edit as below.

```
public class ManageTaskSaga : Saga,
        IAmStartedBy<AddNewTaskCommand>,
        IHandleMessages<UpdateTaskCommand>
{

}
```

Now add some more code in the body of the class to implement the **IHandleMessage** interface.

```
public void Handle(UpdateTaskCommand message)
{
    // Dehydrates all events from event store for given aggregate
    var task = Repository.GetById<Task>(message.TaskId);

    // Triggers the UPDATE-GENERAL event
    task.UpdateModel(message.Title, message.Description, message.DueDate,
                     message.Priority, message.Status);
    Repository.Save(task);

    // Notify back
    var notification = new UpdateTaskNotifyCommand(message.SignalrConnectionId)
    {
        TaskId = task.TaskId,
        Title = task.Title
    };
    Bus.Send(notification);
}
```

- This code won't compile yet as you also need to add an **UpdateTaskNotifyCommand** class in the Commands folder of the **CommandStack** project.

```
public class UpdateTaskNotifyCommand : NotifyCommand
{
    public UpdateTaskNotifyCommand(string connectionId)
        : base(connectionId)
    {
    }

    public Guid TaskId { get; set; }
    public string Title { get; set; }
}
```

- Finally, you also need to add a new **UpdateModel** method of the Task class in the **Task.cs** file in the Model folder of the **CommandStack** project. Before that, though, also add a **TaskUpdatedEvent** class in the Events folder of the Shared project.

```
using System;
using Memento;

namespace TaskZero.Shared.Events
{
    public class TaskUpdatedEvent : DomainEvent
    {
        public TaskUpdatedEvent(Guid id, string title, string description,
                                DateTime? dueDate, Priority priority, Status status)
        {
            TaskId = id;
            Title = title;
            Description = description;
            DueDate = dueDate;
            Priority = priority;
            Status = status;
        }

        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; }
    }
}
```

Now add the following the following **UpdateModel** method to the Task class.

```
public void UpdateModel(string title, string description, DateTime? dueDate,
                        Priority priority, Status status)
{
    var updated = new TaskUpdatedEvent(TaskId, title, description, dueDate, priority, status);
    RaiseEvent(updated);
}
```

- The **Task** class also needs to be added some code to handle the updated event

```
public class Task : Aggregate,
    IApplyEvent<TaskCreatedEvent>,
    IApplyEvent<TaskUpdatedEvent>
{
}
```

Add also the following code to handle the **IApplyEvent** interface.

```
public void ApplyEvent(
    [AggregateId("TaskId")] TaskUpdatedEvent theEvent)
{
    // No need to change TaskId

    // Copy values over
    Title = theEvent.Title;
    Description = theEvent.Description;
    DueDate = theEvent.DueDate;
    Priority = theEvent.Priority;
    Status = theEvent.Status;
}
```

- This is enough to update the Command stack. Let's edit the Query stack as well. Open the **Denormalizers** folder in the ReadStack project and pick up the **ManageTaskDenormalizer** class.

```
public class ManageTaskDenormalizer :
    IHandleMessages<TaskCreatedEvent>,
    IHandleMessages<TaskUpdatedEvent>
{
}
```

- In the same file, also add an implementation for the **IHandleMessage** interface.

```
public void Handle(TaskUpdatedEvent message)
{
    using (var context = new TaskContext())
    {
        var task = (from t in context.PendingTasks
                    where t.TaskId == message.TaskId
                    select t).SingleOrDefault();
        if (task == null)
            return;

        task.Title = message.Title;
        task.Description = message.Description;
        task.DueDate = message.DueDate;
        task.Priority = message.Priority;
        task.Status = message.Status;
        if (message.Status == Status.Completed)
        {
            task.CompletionDate = DateTime.Today;
        }
        if (message.Status == Status.InProgress &&
            task.Status != Status.InProgress)
        {
            task.StartDate = DateTime.Today;
            task.CompletionDate = null;
        }

        context.SaveChanges();
    }
}
```

- The final step consists in notifying back the user interface of the changes on the server. Open the **NotificationHandler.cs** file and add the following:

```
public class NotificationHandler :
    IHandleMessages<AddNewTaskNotifyCommand>,
    IHandleMessages<UpdateTaskNotifyCommand>
```

```
{
}
```

- Now add some code to implement the **IHandleMessage** interface.

```
public void Handle(UpdateTaskNotifyCommand message)
{
    // Notify back
    var hub = new TaskZeroHub(message.SignalrConnectionId);
    hub.NotifyResultOfUpdateTask(message.TaskId, message.Title);
}
```

- All done. Your application now should be able to pick and edit an existing to-do item.