

Common issues

Last updated by | Aleksey Glukharev | 17 May 2025 at 02:20 GMT+4

Performance

The performance is important for this project. This applies to memory, CPU, database and any other remote services or resources.

Avoid making any network requests inside of loops. Database queries are also network requests.

Avoid loading entire entities if all you need is to read a few columns, use projections instead. The only time we should be loading an entire entity (which will be tracked for changes) is when we intend to modify it. In all other cases map the data to an anonymous or a DTO object. When mapping, do not map entire entities to its properties, i.e. don't do `.Select(x => new { x.Ticket })`, as Ticket would be an entity and will be tracked for changes, even though we mapped the outer result to an anonymous object.

Also, not performance related, but some developers tried to reuse the existing DTO and only fill some fields of it. Please don't do this and create a new DTO for your new method instead. Filling only some fields will cause issues later when we accidentally rely on a field that's never actually filled; and if we choose to fill all fields for both cases it will probably lead to code duplication and wasted performance. Oh I guess it is performance related after all.

Please try to always use a single object as an input parameter for all API requests. Even if you need to pass a single value for now. We often need to extend methods in the future and adding new properties to the object is harmless to the existing API calls while adding new arguments to the API method is not (in our case for this framework).

Similarly, please try to always return an object of some type as a result of API calls so that we can extend them later if needed without harming existing API calls.

You do not have to make calls to `repository.Update` or `UpdateAsync` manually to save the changes for each entity. `Update` method attaches the provided entity in a 'modified' state to the context (and doesn't actually save the changes), which in almost all cases is not needed since the entity is already received from the same context and is already being tracked for changes. All entities that are tracked for changes get saved automatically on exiting the API method. You can manually call `CurrentUnitOfWork.SaveChangesAsync()` if needed, but don't do it unless it's necessary.

Commit messages and branch name

When you create a branch name or write a commit message, try not to think in terms of "what am I doing", which sometimes leads to bad messages and names like "fix", "fixed quote", "updated logic", etc. **Instead**, think of it in terms of "how is my branch or commit different from all other branches or commits that were made for the same exact file(s)". If we were to see the commit history for the single file, would it be helpful if all we saw was "Fixed quote", "Fixed quote", "Fixed quote"? You can also see how bugs are not captioned "Bug in quote", "Bug in quote", "Bug in quote", but instead are a lot more descriptive, so that when we see 3 bugs created and even if all 3 are related to quotes, we can distinguish them just by looking at their title, without having to open them and read all the repro steps. The commit messages and branch names should be done the exact same way. We don't need to have to open each single commit and spend time reading all the changes to be able to know what was done. We should be able to know the gist just from the commit message alone. We should be

able to do it both when seeing the history of a single file, and when looking at the commit history in general, regardless of a specific file.

Examples of good commit messages:


```
#14416 Populated OrderLine.QuoteServiceId when creating an order from a quote
#12349 Added 'Tax Exempt' readonly checkbox to order views
#14383 Added 'HideLoadAtAndDeliverToOnHourlyInvoices' setting
#14381 Sorted order summary report by first time on job
#14422 Fixed pagination issue on Organization Units Screen
```

It is often a good idea to split a big task into several atomic commits and provide separate description for each portion of the work that was done.

Even when added for the same bug or work item, the commit messages don't have to repeat the bug title and should describe what portion of the work you did:



```
#13651 Allowed to add ticket photo on Edit ticket modal opened from invoice
#13651 Allowed to add or delete ticket photo regardless of it being on pay statements or invoiced
#13651 Fixed dropdown fields rollback logic on Tickets by Driver view
#13651 Disallow editing some values on Tickets by Driver view based on quoteId
#13651 Made Customer editable, added prompts when editing Customer or PO Number on Tickets by Driver view
```

The above series of commits were done for the same work item  **13651 Changes to "Tickets by Driver"**

 Done and you can see how neither commit is labeled "Changes to Tickets by Driver" and instead describes what specifically was done.

or

```
#14383 Added 'Invoice 6' template
#14383 Hidden Load At and Deliver To on hourly invoice lines when setting is true
#14383 Added 'HideLoadAtAndDeliverToOnHourlyInvoices' setting
#14383 Renamed 'Invoice6' template to 'MinimalDescription'
```

This series of commits was done for task  **14383 Customize invoice for JMT**  Done and you can again see how commit messages allow us to see what specifically was done in each commit, and neither is called "Customized invoice for JMT". Separating the single task into separate atomic commits like these makes it much easier to review and follow the changes (for both the author reviewing their own changes before staging and committing, and the reviewers of the PR, and then, sometimes, for a person who is later looking for a bug in those changes, which might be you again).

Each commit is also atomic and does not combine two things into one, i.e. neither commit has to be called "`#14383 Renamed 'Invoice6' template to 'MinimalDescription' and Added 'HideLoadAtAndDeliverToOnHourlyInvoices' setting and Added 'Invoice 6' template and Hidden Load At and Deliver To on hourly invoice lines when setting is true`" - where everything is crumpled into a single commit even though there are clearly several different sets of changes that can each be separated into their own commit, even if done for the same task.

Even if you have already done more work than could fit into an atomic commit, you can still split those changes in your mind, think of which atomic commits this could have been, and then partially stage those specific changes, create the first atomic commit, then stage the next part, create the next atomic commit, and so forth. But of course it is easier if you plan ahead and think of the committing time right from the start. Before you start writing your code, try to find and isolate subtasks from the task that was given to you, and do one atomic commit worth of changes at a time (then review, stage, commit, and repeat until the whole task is done).

Staging

Please manually stage the specific parts of the code that you have already actively reviewed and want to be merged into main. Avoid submitting meaningless changes like these in the middle of the file as it distracts us during the review process when you submit your PR, and then again when someone fixes it after you and submit their PR and the formatter fixes it back:

<pre> 422 public int OrderLineId { get; } 423 public string TruckCode { get; } 424 public decimal Utilization { get; } 425 } 426 - 427 [AbpAuthorize(AppPermissions.Pages_Orders_Edit)] 428 public async Task SetOrderOfficeId(SetOrderOfficeIdInput input) 429 { 430 var order = await _orderRepository.GetAsync(input.OrderId); 431 if (order.LocationId == input.OfficeId) 432 { </pre>	<pre> 1470 public int OrderLineId { get; } 1471 public string TruckCode { get; } 1472 public decimal Utilization { get; } 1473 } 1474 + 1475 [AbpAuthorize(AppPermissions.Pages_Orders_Edit)] 1476 public async Task SetOrderOfficeId(SetOrderOfficeIdInput input) 1477 { 1478 var order = await _orderRepository.GetAsync(input.OrderId); 1479 if (order.LocationId == input.OfficeId) 1480 { </pre>
--	---

Code style

In general, please follow the code style that is already present in the repository. In some cases few rules might be different per repository, but most will still match.

However, we didn't always have code review process set up, so some changes were never reviewed. And we're also trying to improve over time so some code might be outdated. Please try to adhere to our suggestions for all newly written code.

JS: async/await or callbacks?

Please try to always use async/await instead of callback whenever possible, especially for the new code. Most confirmation dialogs would have an async response and won't require you to pass a callback, you can await network requests, etc. Also, use `try { } catch { } finally { }` instead of `.done().always().catch()` for those async functions as we find them to be much more flexible to work with.

Trailing comma

Please always leave a trailing comma in all (new and old) C# code and in all new JS code. This makes it easier to add new lines later without affecting existing lines, leaving git annotate functionality working as intended.

Usings

Please keep usings sorted. For consistency with the existing files in the project, please check the checkbox to keep system usings on top.

|| or && and newlines

When a line with `||` or `&&` operators is too long and you need to split it, please always move `||` or `&&` operator to the next line together with the line it applies to. I.e. please use this:

```

if (somethingA === 1
    || somethingB === 2

```

rather than this:

```

if (somethingA === 1 ||
    somethingB === 2

```

Or for C#, I'd like you to use this

```
.Where(x => x.SomethingA === 1
      || x.SomethingB === 2
```



instead of this

```
.Where(x => x.SomethingA === 1 ||
      x.SomethingB === 2
```



Keeping the operator on the next line together with the line it applies to makes it much easier to know which operator is being used at a glance, without reading each line to the end (and without having to get back diagonally to the top right when already reading the next line). And while less importantly, it also sometimes makes it easier to add new lines with a different sign without modifying the previous unaffected line, e.g. modifying

```
somethingA
|| somethingB
|| somethingC
```



into

```
somethingA
|| somethingB
&& somethingB2
|| somethingC
```



requires only 1 (new) line to be affected and existing lines can remain unchanged since they all stay with their own operators.

Local functions

Please **avoid** local functions in C#. The existing codebase still has these in a few places, but they generally make the code much harder to follow.

Whitespace

Avoid leaving whitespace characters on empty lines. I.e. an empty line between different parts of code is fine but should be of 0 length - , not 8/12/16 space or tab characters like Visual Studio already removes those in most cases, and doesn't add them if "enter" is pressed, unless you start typing on that line. We're asking to leave those lines empty (no space characters on them) because they will be eventually removed by someone's VS, and it distracts us, however slightly, from the review process. We review whitespace changes because some developers might not follow the existing code style of the repository and are changing the existing lines for some reason, and we're trying to avoid that. Since we're reviewing whitespace changes, please also review your whitespace changes that you submit for review (but review the code too). Please try to only submit the changes that you want us to review, including any whitespace changes that you wish to submit, this request is not limited to printable characters.

Existing code improvements

It's usually best to leave code improvements of existing code (including style or performance changes) to a separate PR, so that your main PR is focused on fixing the things or adding the code that is specifically related to your task. Limiting the PR to the requested changes makes it much easier to follow and review, for both you and us. If you notice some areas that you think need improving, please create a Teams post or message Aleksey telling about those and we'll discuss it and create a separate task for that.

Code duplication

Avoid code duplication. Instead of copying and pasting a code block, it's usually best to move it into a new function and reuse.

StringLength

```
public class TaxRateEditDto
{
    public int? Id { get; set; }

    [Required]
    [StringLength(50)]
    public string Name { get; set; }
```

In most cases we want to use `EntityStringLengths` instead of hardcoding the length value to make it easier to change the length and void issues where a different length can be specified across `{Entity}`, `{Entity}EditDto` and `CreateOrEdit{Entity}Modal.cshtml`. Instead of the code shown above, we should add a single length to `EntityStringLengths` and then use `EntityStringLengths`'s value from all three places.

Tracking for changes

See the comment in this PR to learn when we need and when we don't need to manually specify that we don't want to be tracking entities for tracking. Short version is "we don't need to call `.AsNoTracking()` when our query does not return any entities."

https://dev.azure.com/wallinginfosystems/DispatcherWeb/_git/DispatcherWeb/pullRequest/1901#1716465164