General tips

* Remove **empty spaces**
* Remove **unused stuff** (usings, params, vars...)
* Remove **useless comments**
* Careful at**modifiers**
* Give appropriate **naming**to everything(naming conventions)

Coding

* group code in regions   **Fields and props**,   **Constructors**,  **Member overrides**,   **private methods**,   **Group by interface**,   **Commands**
* use the **most generic interface** possible, everywhere (Iviewcontainer, IEnumerable instead of entity collection...)
* use of **implementation instead of interface**
* **do not instantiate stuff**always resolve instances when possible, sometimes is better to add the container as parameter in the constructor even if its a anti pattern, we'll use lazy loading so it's a performance compromise we have to take
* no **cases and switches** replace with polymorphism(or smth else)
* in case smth suspicious like **too many trips to the DB** use the profiler
* method contains to **many conditions**, recursion
* method does **more than 1 thing**
* method with **more than 4-5 params**
* **Declare data member instead of prop**if not used on the outside
* possible **Null reference exception**
* **do not hard code**anything except commands unique names

**LL**

* careful not to map the**same entity twice**
* **add branches at the end**
* bring only**required fields**(in any prefetch)
* Careful when setting the exclude/include fields to set the companyFields.Add(CompanyFields.Name);use the company fields from Ncoi.Olytmpus.HelperClasses

SQL

* Check the**joins inner, left**
* **too many columns**, not enough,
* all entities should have the**attached metadata,**
* pune la**FK name si ID**(pt grid filters- verifica daca ii nevoie)
* cand faci view pune **AS** cu numele care se foloseste si in alte views(ex company.name as CompanyName)

View

* Host text formats: **HistoricalTeacherTaskEditHostTextFormat** - Edit Historical Teacher Task [{0}],  **HistoricalTeacherTaskOverviewHostTextFormat** -  Historical Teacher Task [{0}] Overview, **ProjectTeamListHostText** - Project Teams
* use directly **2 level binding source** the entity.property.propertyx if it's only used in the view, if it is used in other places custom prop have to be used return this.ExamSubscription != null && this.ExamSubscription.Exam != null ? this.ExamSubscription.Exam.ExamName : string .Empty;
* (**Specific interface**) register the main view(the out of the box one ModelEditPresenter..) presenter(), such that the special ones get resolved nicely        myContainer.RegisterType< IHistoricalTaskReadOnlyPresenterBrief, HistoricalTaskReadOnlyPresenterBriefBrief >()          myContainer.RegisterType<IModelReadOnlyPresenter <HistoricalTeacherTaskEntity >, HistoricalTeacherTaskReadOnlyPresenter >();
* **(specific Interface)**when opening a specific view interface from a specific list use the specific readonly view interface:                   MyController.ShowReadOnlyViewCommand.Execute( new ModelPayload<HistoricalTeacherTaskEntity >(selectedEntity, new PresenterLoader(typeof (IHistoricalTasksBriefListPresenter ))));
* **(specific interface)**  in the presenter using a custom view override CreateView  protected override void CreateView() {this.View = this.MyContainer.Resolve< INewProjectEditView >();}
* **(specific interface)** if custom controller is used inject it in the presenter constructor
* take care when creating **lookup layouts(...),** take care that the modelName of the used instance is completely prefetched in the EditPresenter(tip: also look in each composing element of the module name it may be a custom prop and may need additional objects in the prefetch)

List View

* check list columns names if they're ok they maybe off sometimes(also the values if a systype enum is used)

Edit view

* ups padding on text boxes and other elements to allow space 20-25

Presenters & Controllers

* **never check validators in the controller**(CanExecute...) check only if arg !=null and arg.Item!=null the check is made when executing (executesavecommand - ModelController thus the button is always active and in case of errors the generic exception handler can display messages - set in the service in checkPreconditionsOnExecuting)
* some **presenters can be called from others** thus additional behavior can be added
* use Presenter loader to create a new presenter, because view name resolution if loader is used an existing presenter can be set to active in case someone wants to open it again opposed to directly calling presenter.Open()
* when **injecting a small presente**r in a large one set the PP of the large one to include the necessary fields of the little one

Services

* In **check preconditions** don't check anything that might need a refresh No properties that can change on refresh(eg: if the service should be enabled for a certain flag and that flag was changed in the meantime and I'm stuck)
* When **checking preconditions on executing** get the preconditions result from base :   var result = base.CheckPreconditionsOnExecuting(serviceArgs);
* **When writing tests** one should name them after the method actually tested like CheckPreconditionsOnExecuting.. **InnerExecute\_** ..(not just Execute\_...)
* test if **any of the dependencies are that the service is using are missing** check what exception it should throw
* test for **affected entities**
* test for **no of times called**