Hierarchy Tree:

```
<CTreeView @ref="tree" TEntity="OrganUnit" TextFunc="t => t.Title"
      ParentNodeFilterFunc="t => t.ParentOrganId == null" FilterFunc="t =>
      (!organSearch.HasValue() || t.Title.StartsWith(organSearch)) &&
      t.ActiveType == activeType">
    <Template>
        <TreeNodeTemplate Levels="new byte[]{1, 2, 3, 4, 5, 6}">
            <span class="t-grid-delete">
            <span style="color:#0dcaf0" @onclick="async () => await
                ShowEditForm(context, false)" class="t-icon t-delete fa fa-
                plus"></span>
            </span>
            <span style="padding:0" class="t-grid-edit" @onclick="async () =>
                await ShowEditForm(context, true)">
                 <span class="t-icon t-edit fa fa-pencil"></span>
            </span>
            <span style="padding:0" class="t-grid-delete"
                 @onclick="async () => await DeleteOrganUnit(context)">
                 <span class="t-icon t-delete fa fa-trash"></span>
            </span>
         </TreeNodeTemplate>
    </Template>
</CTreeView>
```

Autocomplete Single Select:

```
<AutoCompleteTree @bind-Value="value">
   <CTreeView TEntity="OrganUnit" TextFunc="t => t.Title"
        ParentNodeFilterFunc="t => t.ParentOrganId == null"
        FilterFunc="t => (!context.HasValue() ||
        t.Title.StartsWith(context))"/>
 </AutoCompleteTree>
```

Autocomplete Multiple Select:

```
<AutoCompleteTree @bind-Value="values">
    <CTreeView TEntity="OrganUnit" TextFunc="t => t.Title" Selectable
        ParentNodeFilterFunc="t => t.ParentOrganId == null"
        FilterFunc="t => (!context.HasValue() || t.Title.StartsWith(context))"/>
</AutoCompleteTree>
```

```razor
@code
{
    int[] values;
    int? value;
    //----------------------
    ActiveType activeType = ActiveType.Enable;
    string organSearch;
    WindowStatus status;
    OrganUnit organUnit;
    CTreeView<OrganUnit> tree;
    IControl firstControl;

    async Task DeleteOrganUnit(TreeViewItem node)
    {
        var organUnitId = Convert.ToInt32(node.Value);
        using var scope = CreateScope();
        var service = new OrganUnitService(scope.ServiceProvider);
        var organUnit = await service.SingleAsync(organUnitId);
        var result = await service.ValidateRemoveAsync(organUnit);
        if (result.IsValid)
        {
            await service.RemoveAsync(organUnit);
            await service.SaveChangesAsync();
            tree.RemoveFromTree(organUnit);
        }
        else
            ShowMessage(result.Errors.First().ErrorMessage);
    }

    async Task ShowEditForm(TreeViewItem nodeView, bool isUpdate)
    {
        if (isUpdate)
        {
            ///در حالت ویرایش
            var organUnitId = Convert.ToInt32(nodeView.Value);
            using var scope = CreateScope();
            organUnit = await new OrganUnitService(scope.ServiceProvider)
               .SingleAsync(organUnitId);
        }
        else
        {
            organUnit = new OrganUnit();
            organUnit.ActiveType = ActiveType.Enable;
            organUnit.ParentOrganId = nodeView == null ? null :
                Convert.ToInt32(nodeView.Value);
        }
        status = WindowStatus.Open;
    }
```

```
    async Task<bool> UpsertOrganUnit()
    {
        using var scope = CreateScope();
        var service = new OrganUnitService(scope.ServiceProvider);
        if (organUnit.Id == 0)
            await service.AddAsync(organUnit);
        else
            await service.UpdateAsync(organUnit);
        await service.SaveChangesAsync();
        tree.UpsertInTree(organUnit);
        status = WindowStatus.Close;
        StateHasChanged();
        return true;
    }
}
```