Alternative Caching Approach

...or how to have lazy evaluated, fresh data and 0* sql queries

relaxt confusion labs | **Stephan Bauer**

Feel free to contact me in case you have questions:

GitHub: sb-relaxt-at
Slack: sb
Mail: me@stephan.is

In especially it is a caching approach that creates the cached value on demand (lazy) and ensures non outdated (fresh) results (as long as you don't mess up the configuration) but - in contrast to partial caching - needs no sql queries to check for freshness.

Yet it is not published. We recently ported it to Silverstripe CMS 4 and will check if we can open source it.

NEW PRESTIGE – Luxus Residenzen in Premium Lage
1040 Wien

Fläche: 30 bis 159 m²	**Freie Wohnungen:** 33
Zimmer: 1 - 5	**Preis:** € 1.255.000

THE SECRET GARDEN – Luxus Residenzen am Belvedere
Prinz Eugen-Straße 10 a, 1040 Wien

Fläche: 77 bis 202 m²	**Freie Wohnungen:** 4
Zimmer: 2 - 6	**Preis:** Auf Anfrage

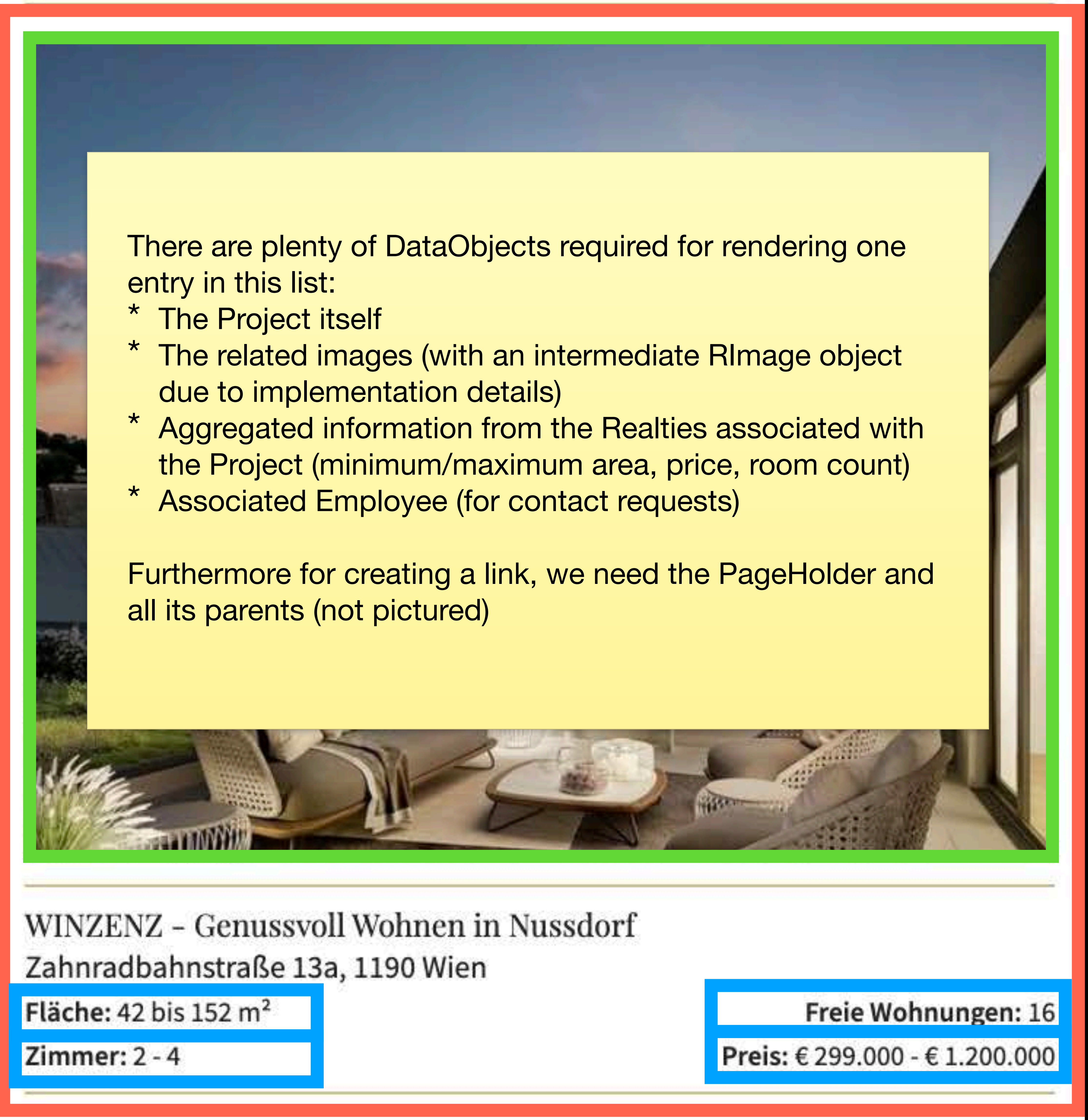
WINZENZ – Genussvoll Wohnen in Nussdorf
Zahnradbahnstraße 13a, 1190 Wien

Fläche: 42 bis 152 m²	**Freie Wohnungen:** 16
Zimmer: 2 - 4	**Preis:** € 299.000 - € 1.200.000

COMING HOME – Heimkommen und Wohlfühlen in Korneuburg!
Schaumannstraße 38, 2100 Korneuburg

Fläche: 37 bis 144 m²	**Freie Wohnungen:** 1
Zimmer: 1 - 4	**Preis:** € 143.500 - € 499.000

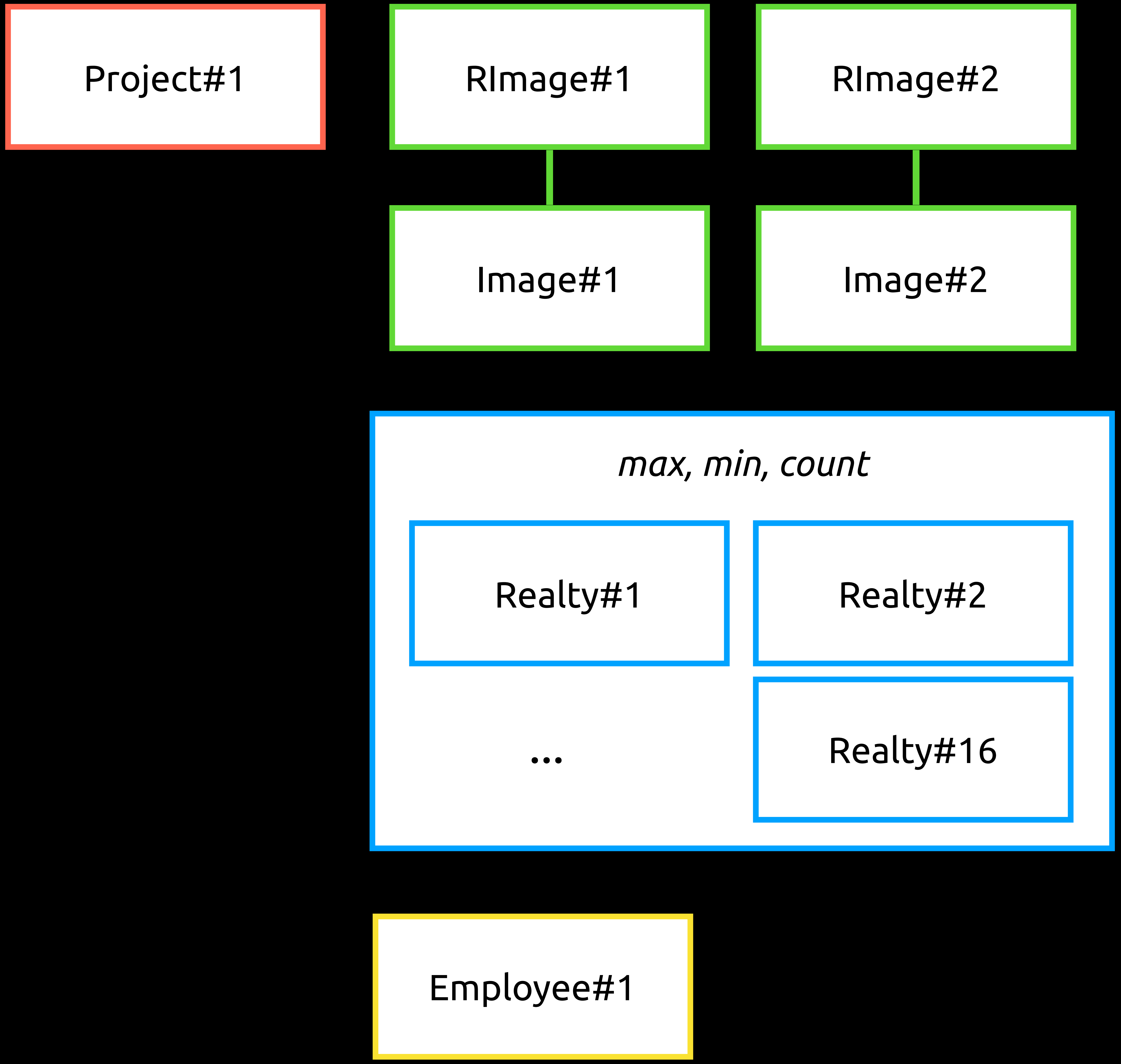
That’s a list of real estate projects from a real web project of a customer.

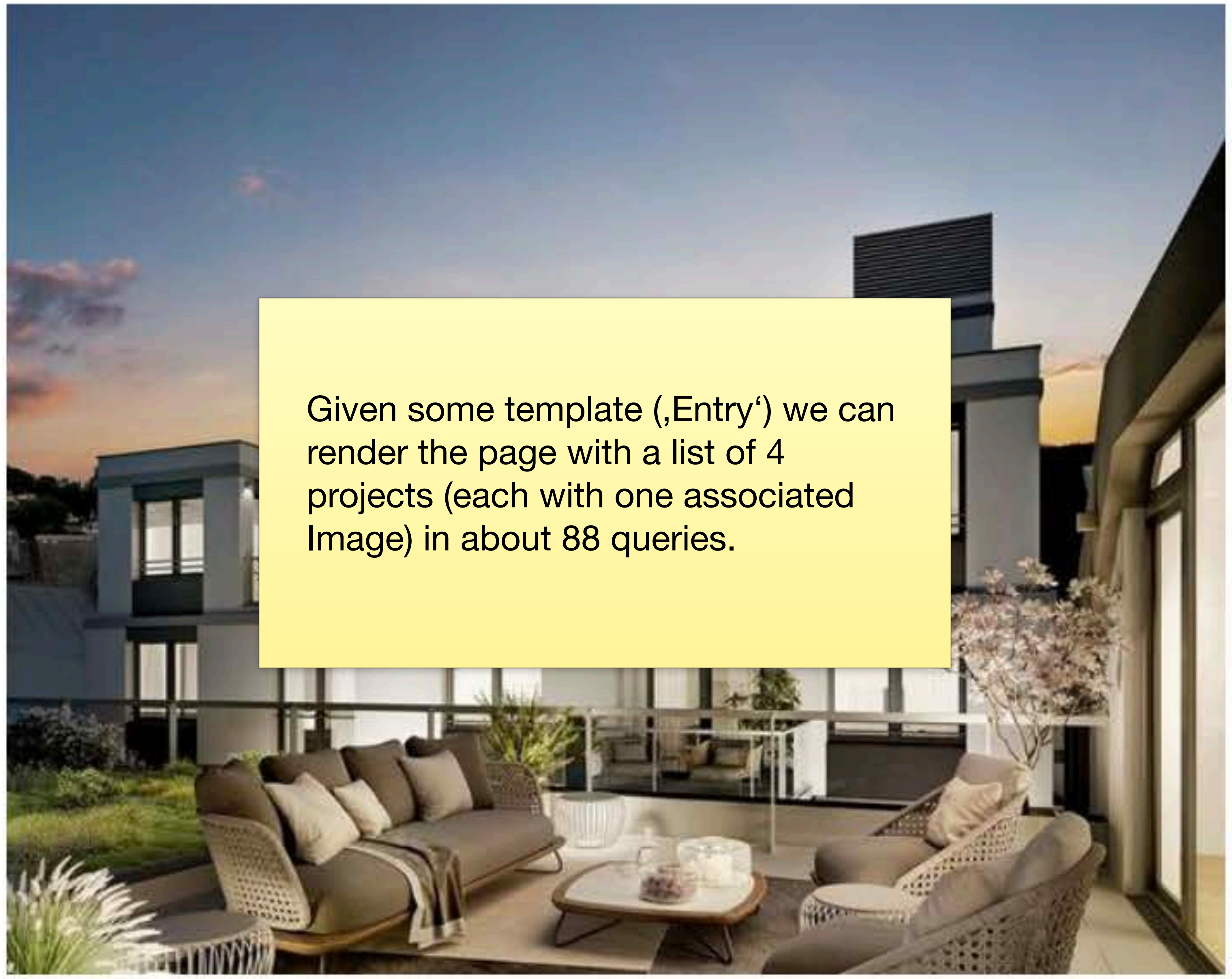


There are plenty of DataObjects required for rendering one entry in this list:
* The Project itself
* The related images (with an intermediate RImage object due to implementation details)
* Aggregated information from the Realties associated with the Project (minimum/maximum area, price, room count)
* Associated Employee (for contact requests)

Furthermore for creating a link, we need the PageHolder and all its parents (not pictured)

WINZENZ – Genussvoll Wohnen in Nussdorf
Zahnradbahnstraße 13a, 1190 Wien
Fläche: 42 bis 152 m²
Zimmer: 2 - 4
Freie Wohnungen: 16
Preis: € 299.000 - € 1.200.000





Given some template (‚Entry‘) we can render the page with a list of 4 projects (each with one associated Image) in about 88 queries.

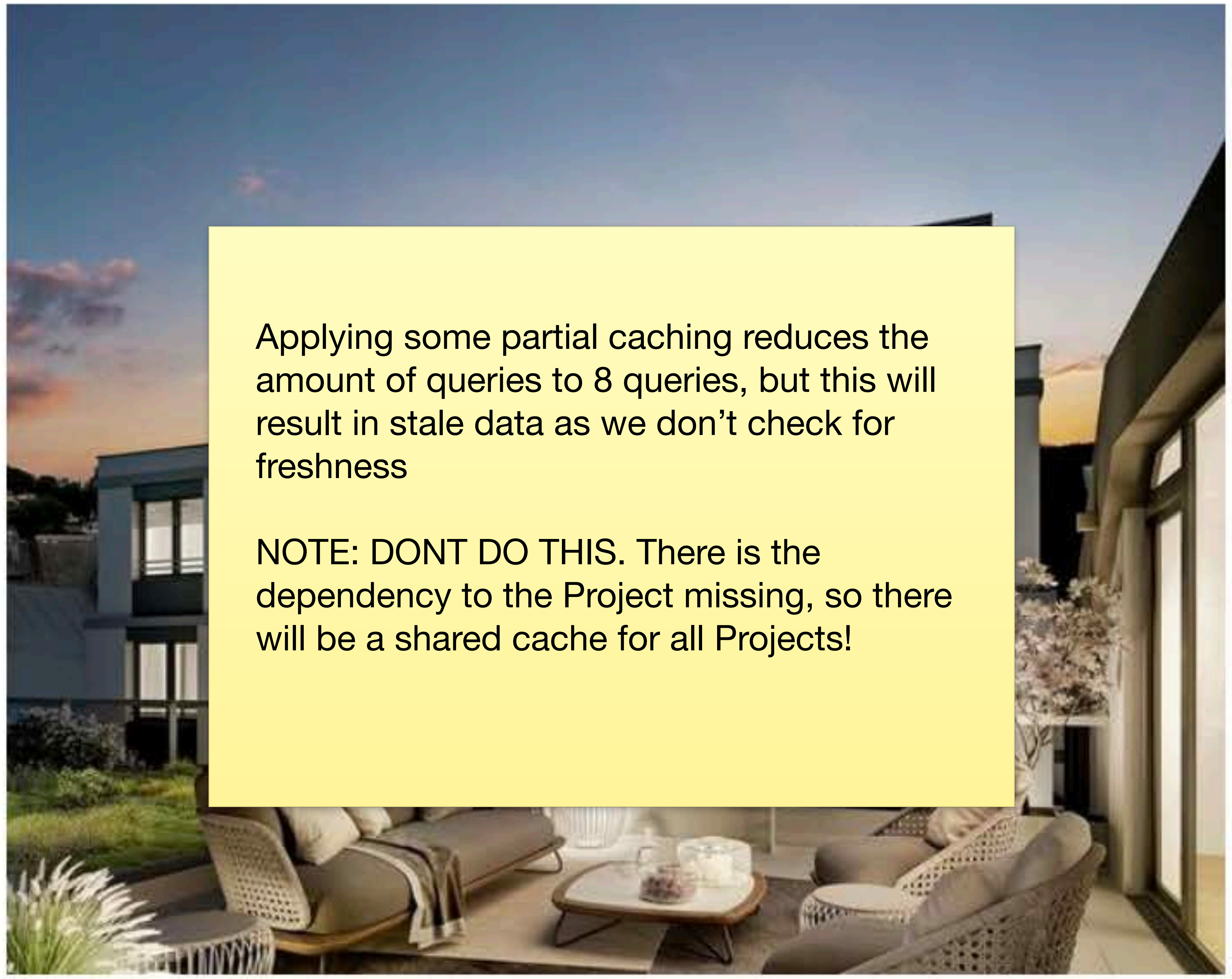
WINZENZ – Genussvoll Wohnen in Nussdorf
Zahnradbahnstraße 13a, 1190 Wien

Fläche: 42 bis 152 m² **Freie Wohnungen:** 16

Zimmer: 2 - 4 **Preis:** € 299.000 - € 1.200.000

```
<% loop $Projects %>
  <% include Entry %>
<% end_loop %>
```

4 Projects | **88 queries**



Applying some partial caching reduces the amount of queries to 8 queries, but this will result in stale data as we don’t check for freshness

NOTE: DONT DO THIS. There is the dependency to the Project missing, so there will be a shared cache for all Projects!

WINZENZ – Genussvoll Wohnen in Nussdorf
Zahnradbahnstraße 13a, 1190 Wien

Fläche: 42 bis 152 m² **Freie Wohnungen:** 16

Zimmer: 2 - 4 **Preis:** € 299.000 - € 1.200.000

```
<% loop $Projects %>
  <% cached 'Entry' %>
    <% include Entry %>
  <% end_cached %>
<% end_loop %>
```

4 Projects | **8 queries**

```
<% loop $Projects %>
    <% cached 'Entry',
        Project.ID, Project.LastEdited,
        Images.Count, Images.max('LastEdited'),
        Contact.ID, Contact.LastEdited,
        Realties.Count, Realties.max('LastEdited'),
        HolderPage.ID, HolderPage.LastEdited,
        HolderPage.Parent.ID, HolderPage.Parent.LastEdited
    %>
        <% include Entry %>
    <% end_cached %>
<% end_loop %>
```

4 Projects | **36 queries (113 queries)**

So let's add the checks for freshness of the cached data. There are several unsolved issues:
* It's a quite verbose syntax
* We cannot check the two leveled relations with RImage and Image easily
* It's difficult to get all parents of the HolderPage

In brackets: The queries needed when the cache is cold.

```
<% loop $Projects %>
    <% cached 'Entry',
        Project.ID, Project.LastEdited,
        Images.Count, Images.max('LastEdited'),
        Contact.ID, Contact.LastEdited,
        Realties.Count, Realties.max('LastEdited'),
        HolderPage.ID, HolderPage.LastEdited,
        HolderPage.Parent.ID, HolderPage.Parent.LastEdited
    %>
        <% include Entry %>
    <% end_cached %>
<% end_loop %>
```

But foremost it doesn't scale well:
* The freshness check requires 8 queries per Project so if you have ten times the Projects you have ten times the queries
* It is not (easily/properly/at all) possible to cache inside and outside of the loop (see https://docs.silverstripe.org/en/4/developer_guides/performance/partial_caching/#nested-cache-blocks)
* And even if nesting would be possible it would be very difficult to define the cache key for the surrounding block.

40 Projects | **~270 queries**

```
<% loop $Projects %>
  {$renderEntry()}
<% end_loop %>
```

```
public function renderEntry(){
  $cachedViewer = CachedViewer::create('Entry', $this);

  $cachedViewer->setCalculateCacheDependency(function(){
    $cacheDependency = new CacheDependency();
    $cacheDependency->require($this,
      '[Images.Image,Contact,Realties,HolderPage.Parent^]'
    );
    return $cacheDependency;
  });

  return $cachedViewer;
}
```

Our approach: We define regions that should be cached in a CachedViewer. We pass the Project ($this) and the template to render (‚Entry‘). In case that there is no cached result, the view is rendered and the dependencies to related DataObjects are calculated. Therefore we need to pass a function that returns a so-called CacheDependency. As we only calculate this CacheDependency when creating the cached result, we don’t need to perform queries when reading.

4 Projects | **8 queries (136 queries)**

```
<% loop $Projects %>
  {$renderEntry()}
<% end_loop %>
```

```
public function renderEntry(){
  $cachedViewer = CachedViewer::create('Entry', $this);

  $cachedViewer->setCalculateCacheDependency(function(){
    $cacheDependency = new CacheDependency();
    $cacheDependency->require($this,
      '[Images.Image,Contact,Realties,HolderPage.Parent^]'
    );
    return $cacheDependency;
  });

  return $cachedViewer;
}
```

This magic syntax passed to the CachedDependeny is similar to the definition in partial caching but much simpler:
* Images.Image requires all the RImage objects as well as the associated Image objects (you can nest as many levels as you want)
* It can be a HasOne, HasMany or ManyMany
* HolderPage.Parent^ is recursively requiring all parent Pages as well

4 Projects | **8 queries (136 queries)**

```
<% loop $Projects %>
  {$renderEntry()}
<% end_loop %>
```

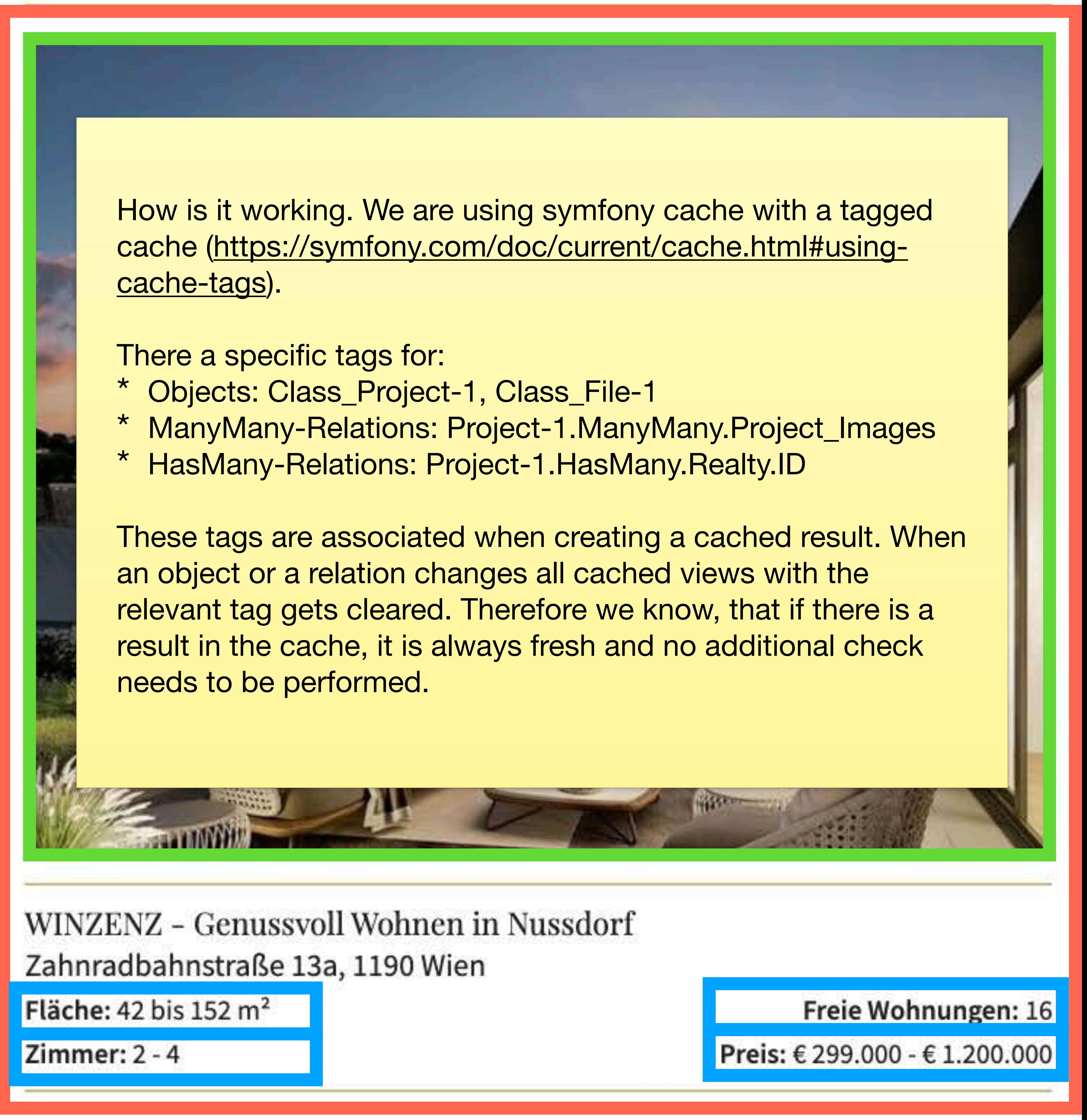
```
public function renderEntry(){
  $cachedViewer = CachedViewer::create('Entry', $this);

  $cachedViewer->setCalculateCacheDependency(function(){
    $cacheDependency = new CacheDependency();
    $cacheDependency->require($this,
      '[Images.Image,Contact,Re
    );
    return $cacheDependency;
  });

  return $cachedViewer;
}
```

As we don’t need to query the database when we have a cache result, it scales well. If we recursively cache the blocks as well, we can reduce it to 4 queries. (Of course on a real page there are other things like a menu, but you could cache those as well)

If you are curious what the remaining queries are (Silverstripe CMS 3):
* 2 queries for resolving the path (path is something like „/projects/list“)
* 1 query for checking it should be on root
* 1 query for fetching the SiteConfig

40 Projects | ~~**8 queries**~~ **4 queries** *(which is almost 0)*

Symfony cache



WINZENZ – Genussvoll Wohnen in Nussdorf
Zahnradbahnstraße 13a, 1190 Wien

Fläche: 42 bis 152 m²
Zimmer: 2 - 4

Freie Wohnungen: 16
Preis: € 299.000 - € 1.200.000

How is it working. We are using symfony cache with a tagged cache (https://symfony.com/doc/current/cache.html#using-cache-tags).

There a specific tags for:
* Objects: Class_Project-1, Class_File-1
* ManyMany-Relations: Project-1.ManyMany.Project_Images
* HasMany-Relations: Project-1.HasMany.Realty.ID

These tags are associated when creating a cached result. When an object or a relation changes all cached views with the relevant tag gets cleared. Therefore we know, that if there is a result in the cache, it is always fresh and no additional check needs to be performed.

```
Class_Project-1
Project-1.ManyMany.Project_Images
Class_RImage-1
Class_File-1
Class_Employee-1
Project-1.HasMany.Realty.ID
Class_Realty-1
Class_Realty-2
Class_Realty-3
…
Class_Realty-16
Class_SiteTree-1
Class_SiteTree-2
```

dynamic content

getting depedencies right

There are some issues that always come with caching:

* If there is dynamic content (random order, forms) we need to handle this somehow
* You need to carefully define the dependencies, but it is a little bit easier when you can cache smaller parts
* When the cache is cold there is an additional performance impact (maybe it would be possible to optimize the existing code)

performance impact when cold

There are some specific issues with this approach:
* We need to overwrite the ManyManyList in order to get notified of changes (might be integrated into the core?)
* There are some modules (e.g. for reordering GridFields) bypassing the ORM which makes it impossible to track changes
* Invalidation takes some time (but in our existing, smaller setups this has not been an issue yet)

overwriting ManyManyList

modules bypassing ORM

invalidating takes time

nested viewers

Requirements

There are two common issues that can be solved by this approach:
* We have recently added support for Requirement call from inside a cached view (either in template or code)
* Nested viewers are yet not automatically supported but it is quite easy to collect the nested viewers, get all their tags and add them to the parent cached viewer.

Feel free to contact me in case you have questions:

GitHub: sb-relaxt-at
Slack: sb
Mail: me@stephan.is