Hello DF23

Nice to meet you

Building a Strong Foundation for Your Modern .NET Project

Roy Berris

Software Engineer at iO

Working with

- Umbraco
- MACH
- .NET

Design fase

We need to design.
We need to think.
We need to discard.
We need to start.

A Strong Foundation

Looking at construction

We're similar to the construction world.

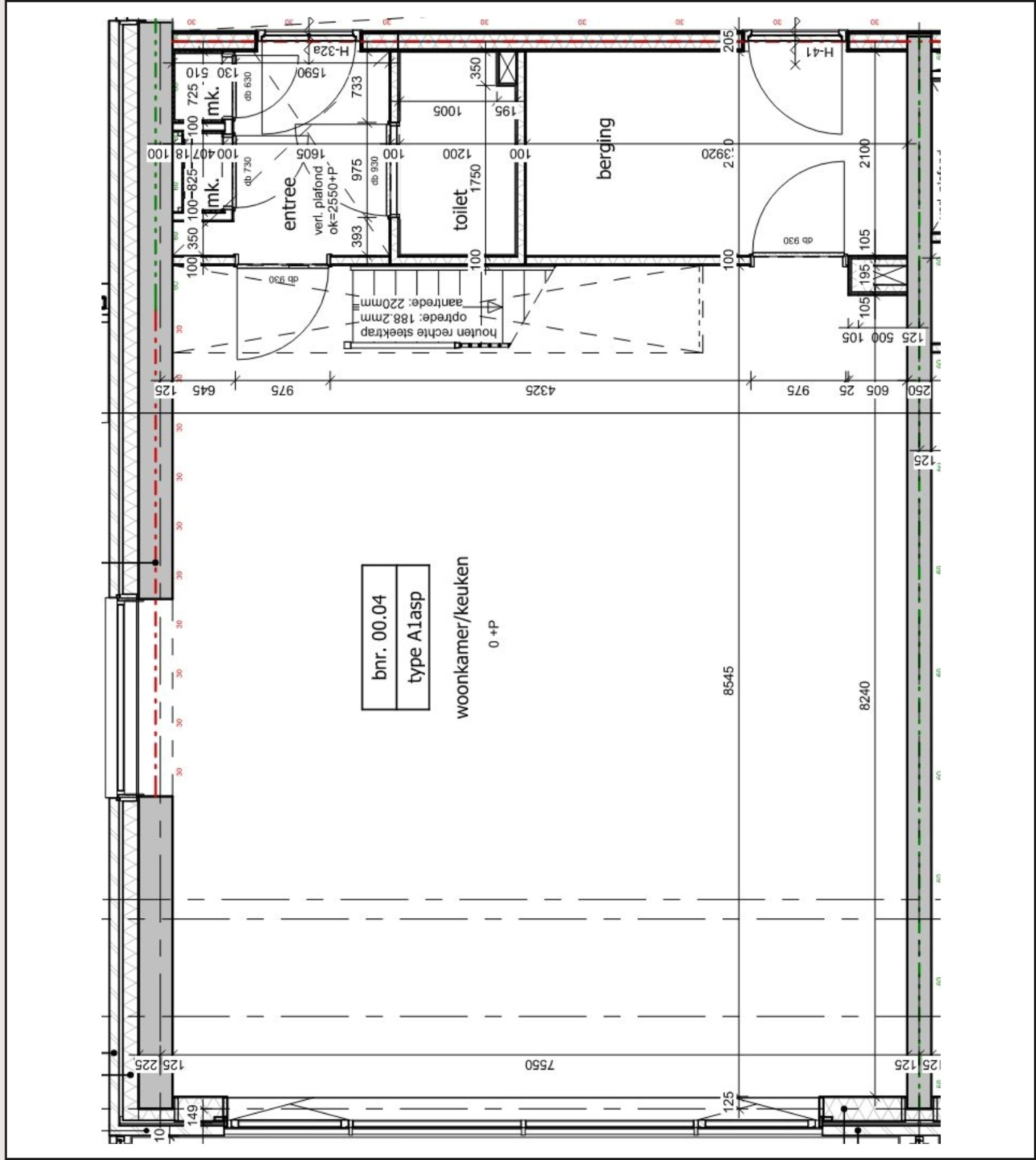
- They plan
- They build
- They maintain

Comparing a floor plan

Create a floorplan that suites your application.

- The doors is for outgoing information
- The windows is for incoming information
- The toilet is our trashbin
- The storage room is our... database

This is our architecture

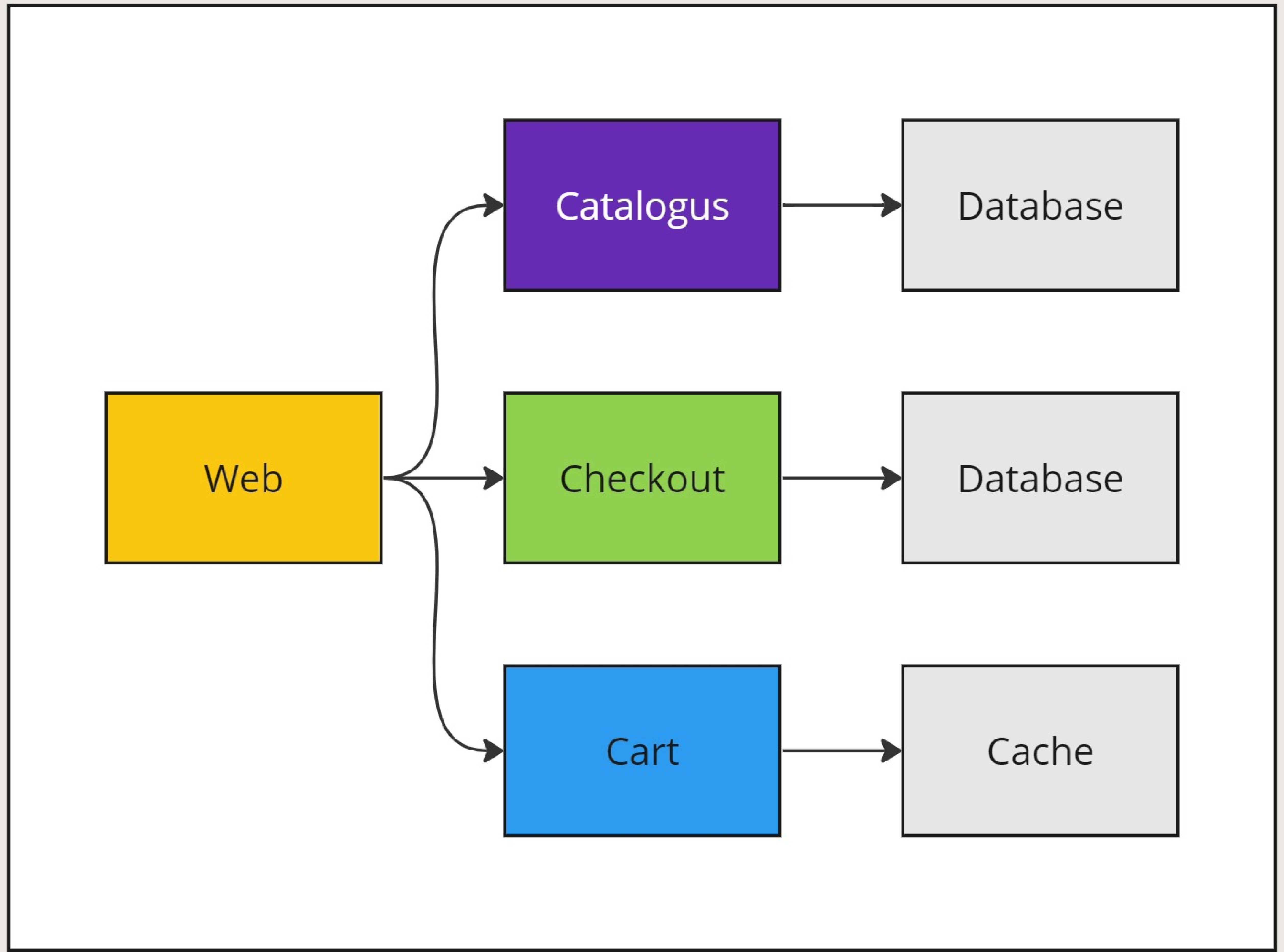


Software architecture vs design

Architecture is the concern of the whole.

Design is the concern of the individual component.

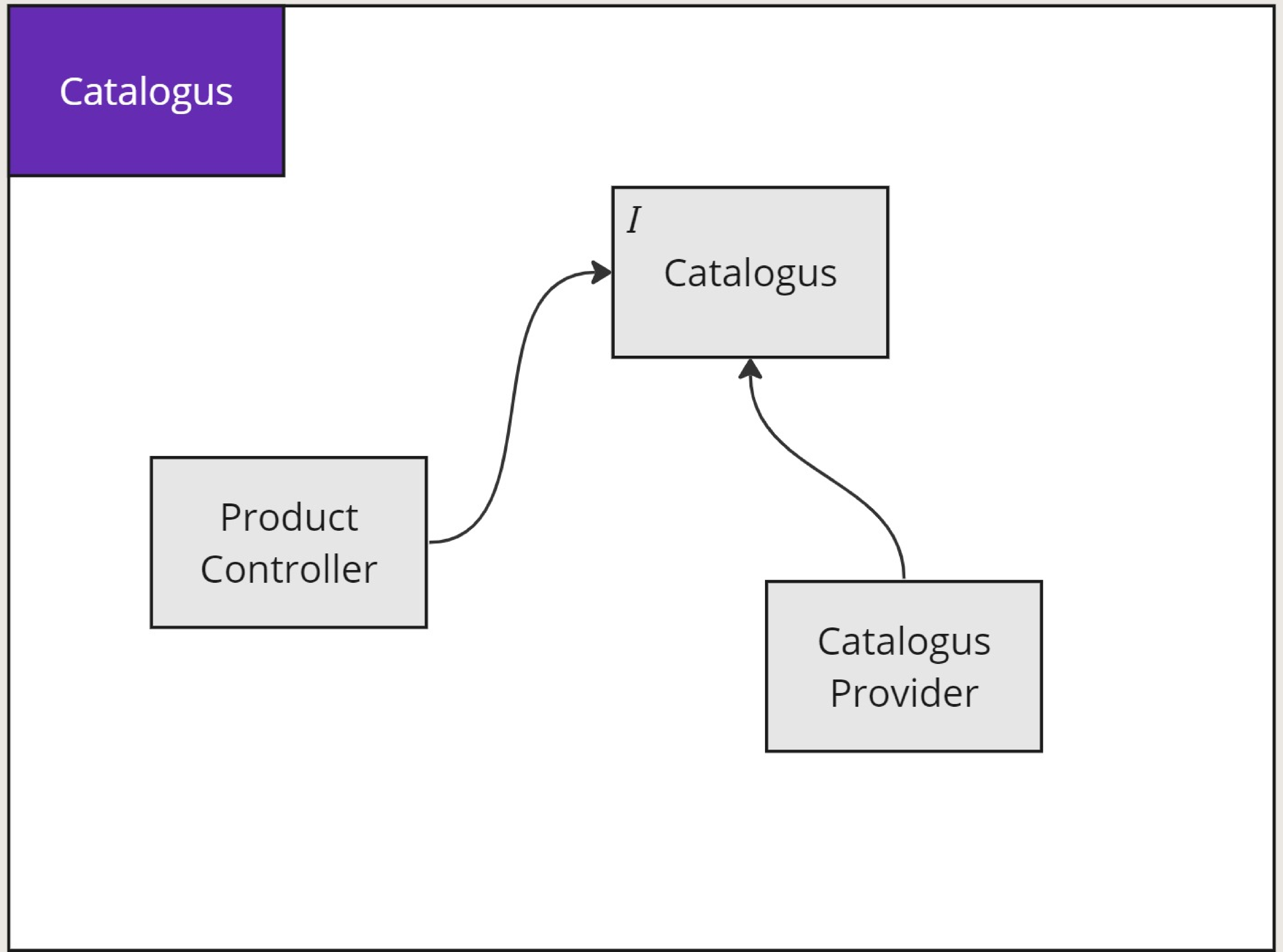
Architecture



A high level abstract overview of the system. We think about:

- Maintenance
- Performance
- Application quality

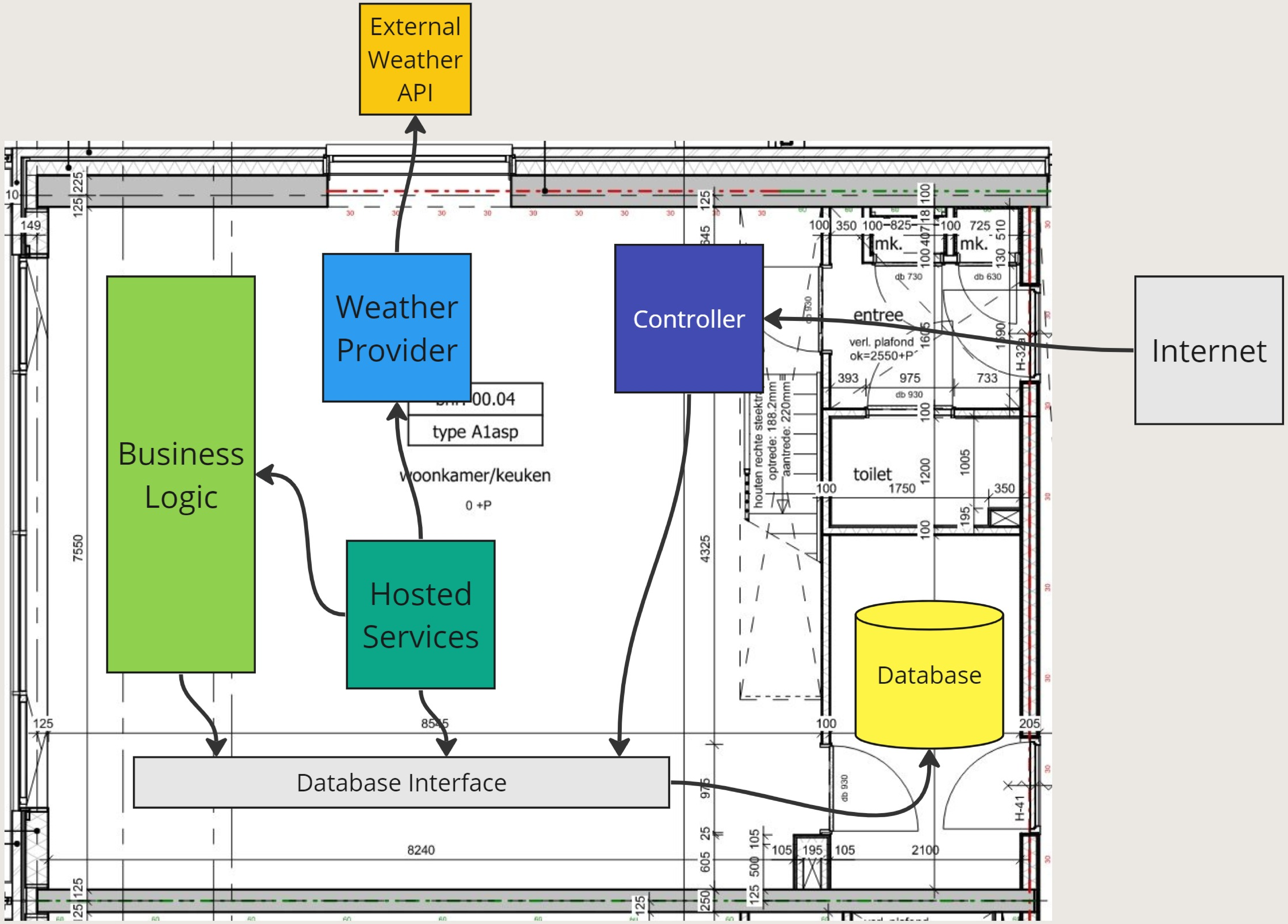
Design



A detailed specification of a software component.

It's a design plan to implement the component.

- Limited to one database
- Just enough space to crush all the business logic
- Only one window to query data



The Wrong Foundation

There is no such thing if

A foundation is never wrong if its purpose is met.

A foundation is 'wrong' when the purpose changes.

When does the purpose change?

The foundation was build for this house. It's purpose is met.

When does the purpose change?

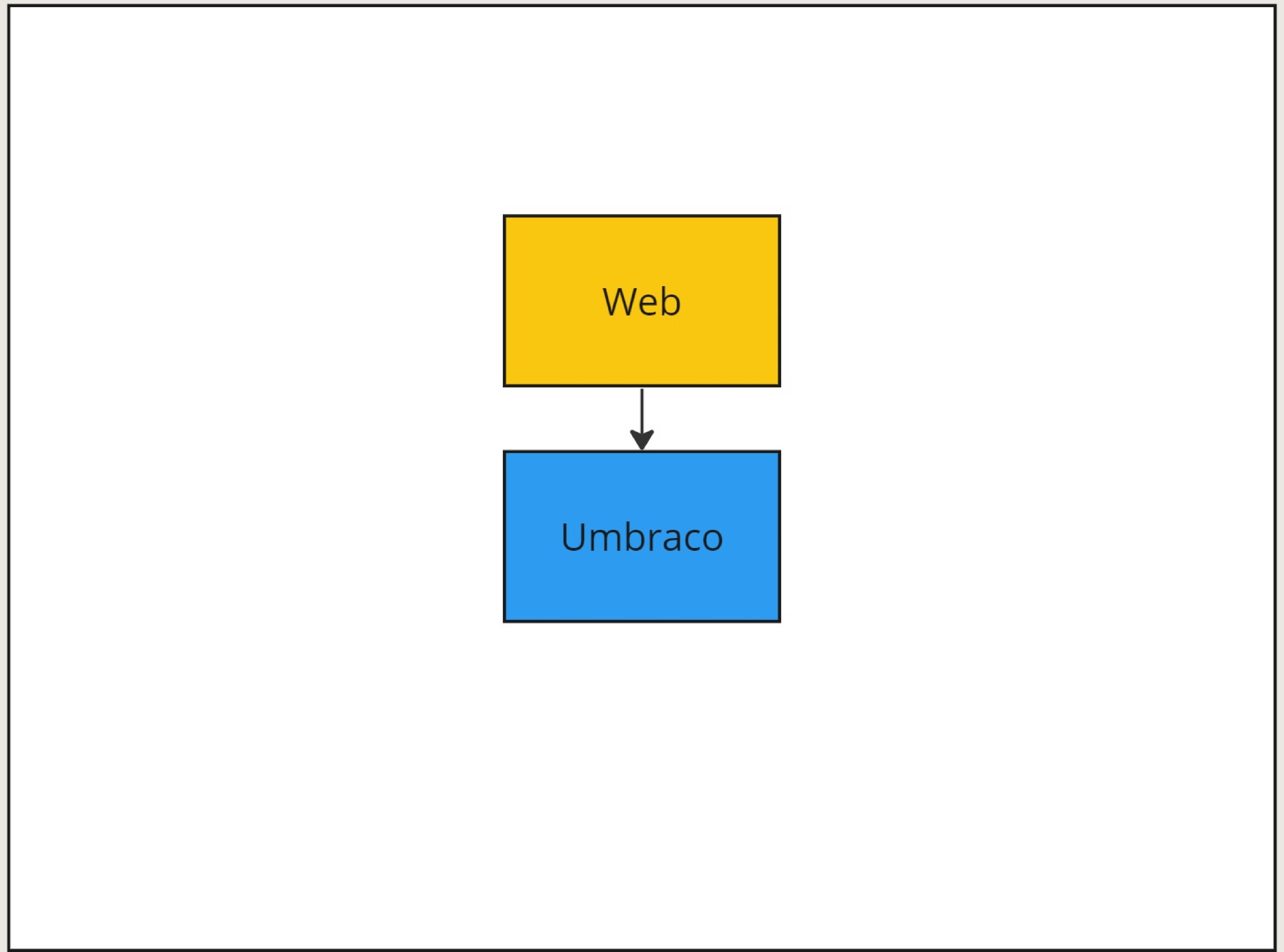
The foundation for this house has to be extended in order to fit the villa.

When does the purpose change?

Does the foundation allow for this skyscraper to stand?

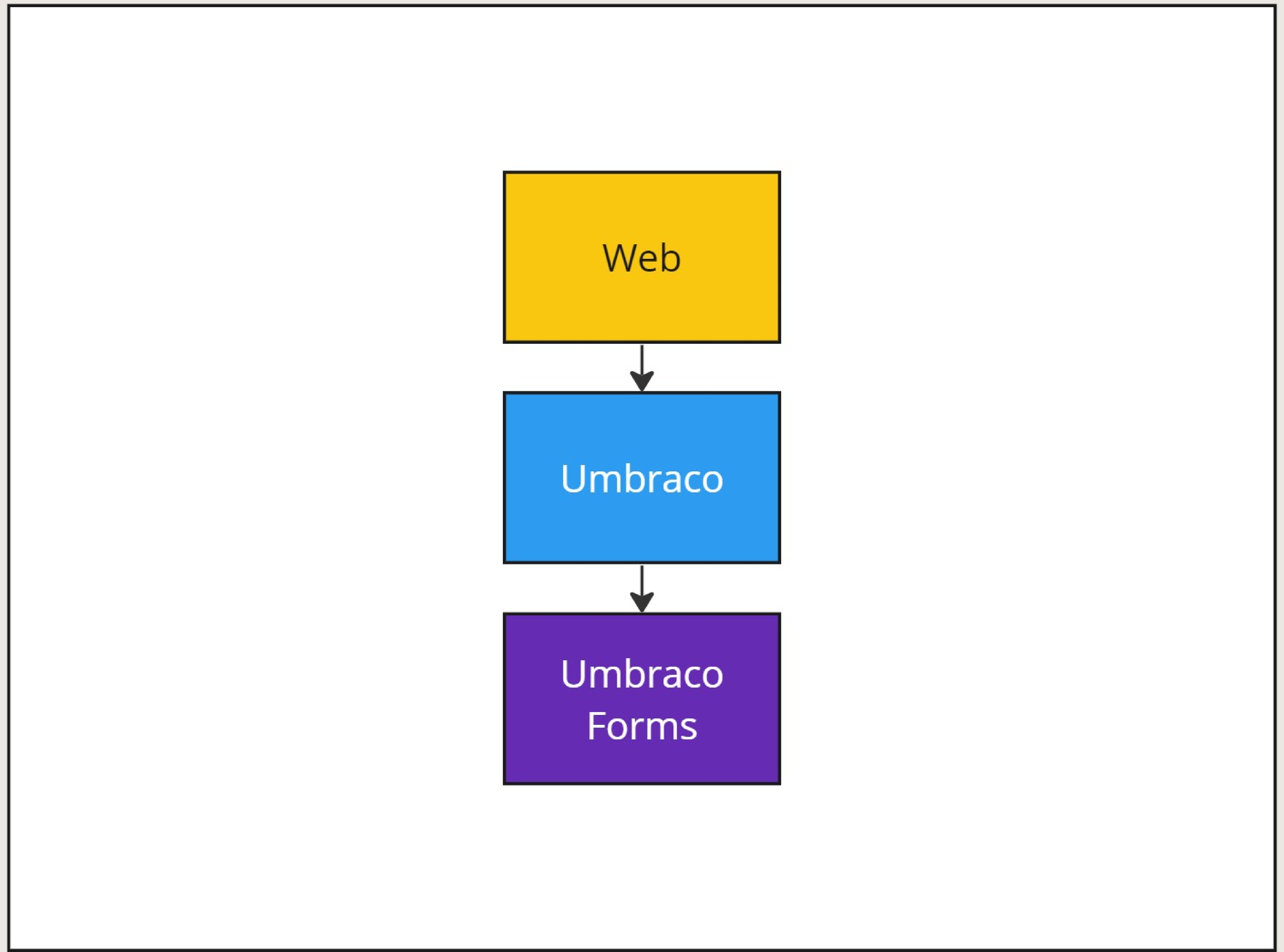
When will it fall in to the ground?

Software is no different



A website for a bakery, producing for the local population. They'll have a simple website with some content, contact information and opening times.

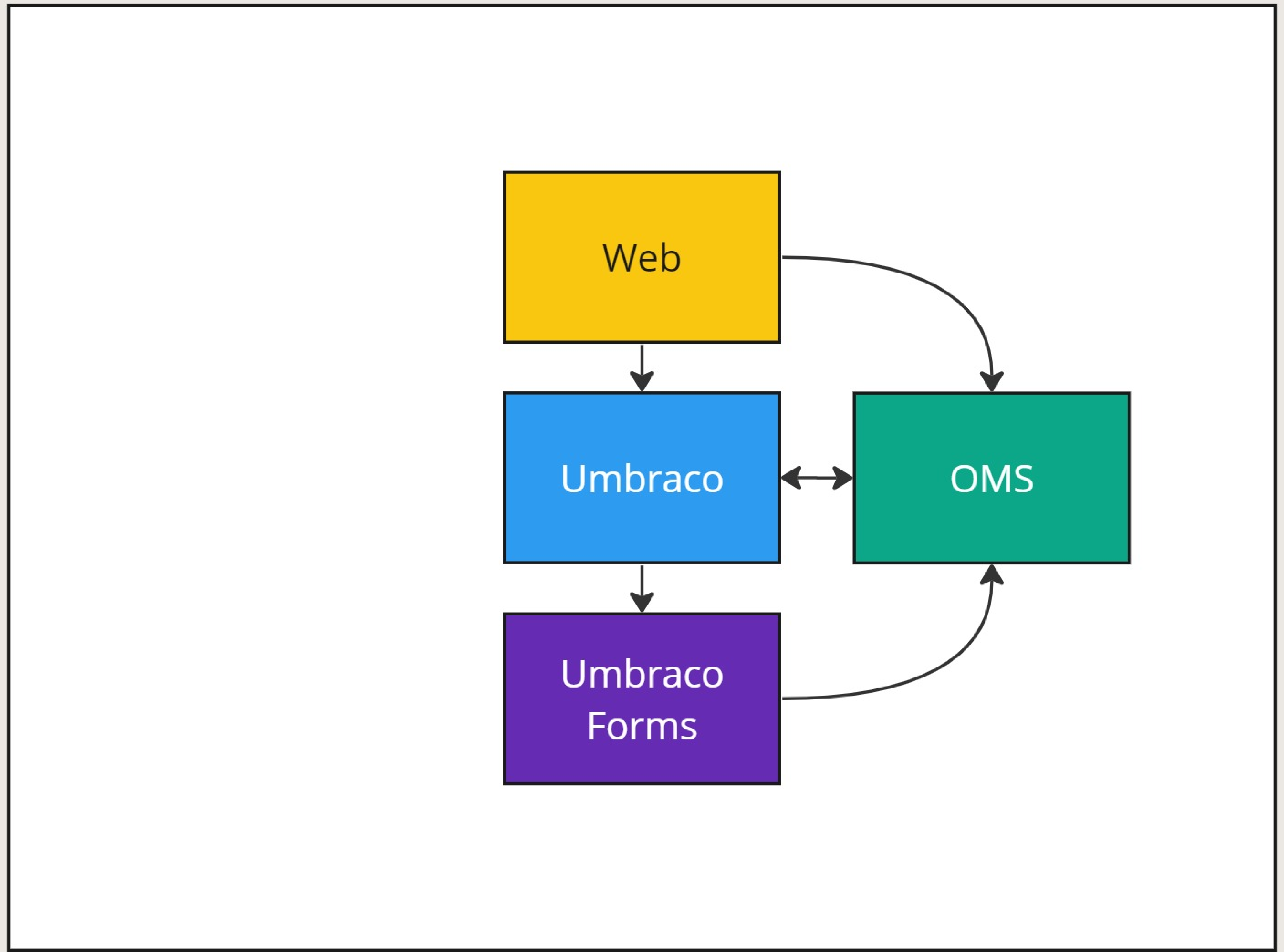
Software is no different



They won an award for their amazing cakes. People are ordering from the bakery like crazy.

They decide to add a ordering system using Umbraco Forms to handle orders by e-mail.

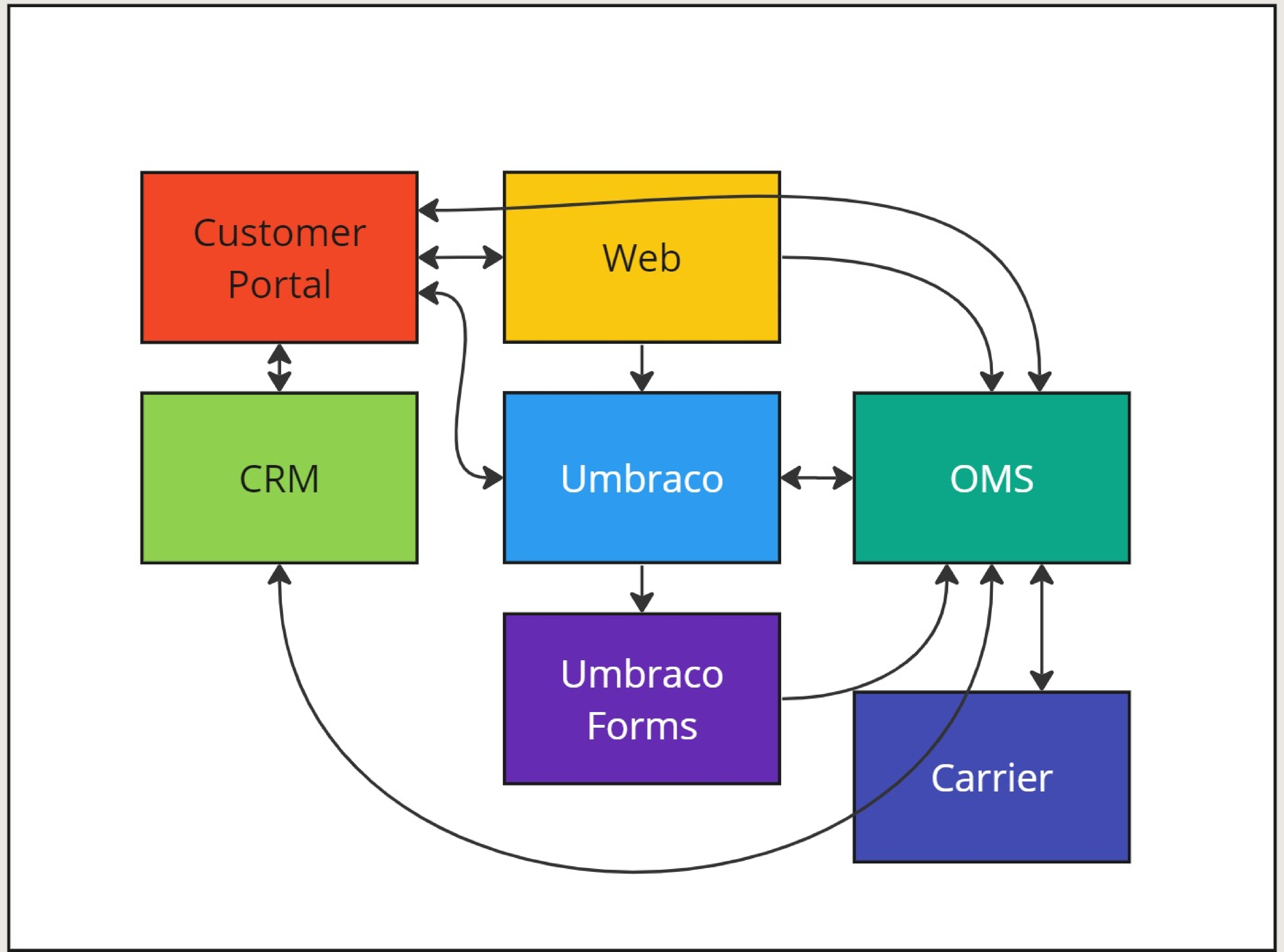
Software is no different



The throughput is too high, Umbraco Forms is no longer practical to handle orders manually. They'll need an automated system and get their products on the website.

They decide to add an OMS which communicates with Umbraco to show the latest delivery times and prices.

Software is no different



The bakery has grown to a company. A lot of supermarkets now sell their cakes. They implement a CRM for sales tracking. They also developed a customer portal so customers can track their orders. They also need to keep their carrier up-to-date on orders.

What happened?

The application became the skyscraper.

The foundation was not wrong, the purpose of the application changed.

It required the foundation to grow with it. And this is not uncommon.

Boom

In the real world the building will collapse.

In our world the application will
- be unstable
- be hard to maintain
- be slow
- be resource intensive
- fail when one module fails

You'll end up with an architecture if you want to or not.

If you don't choose it, it won't fit.

Choose your architecture

Foundational Guidelines

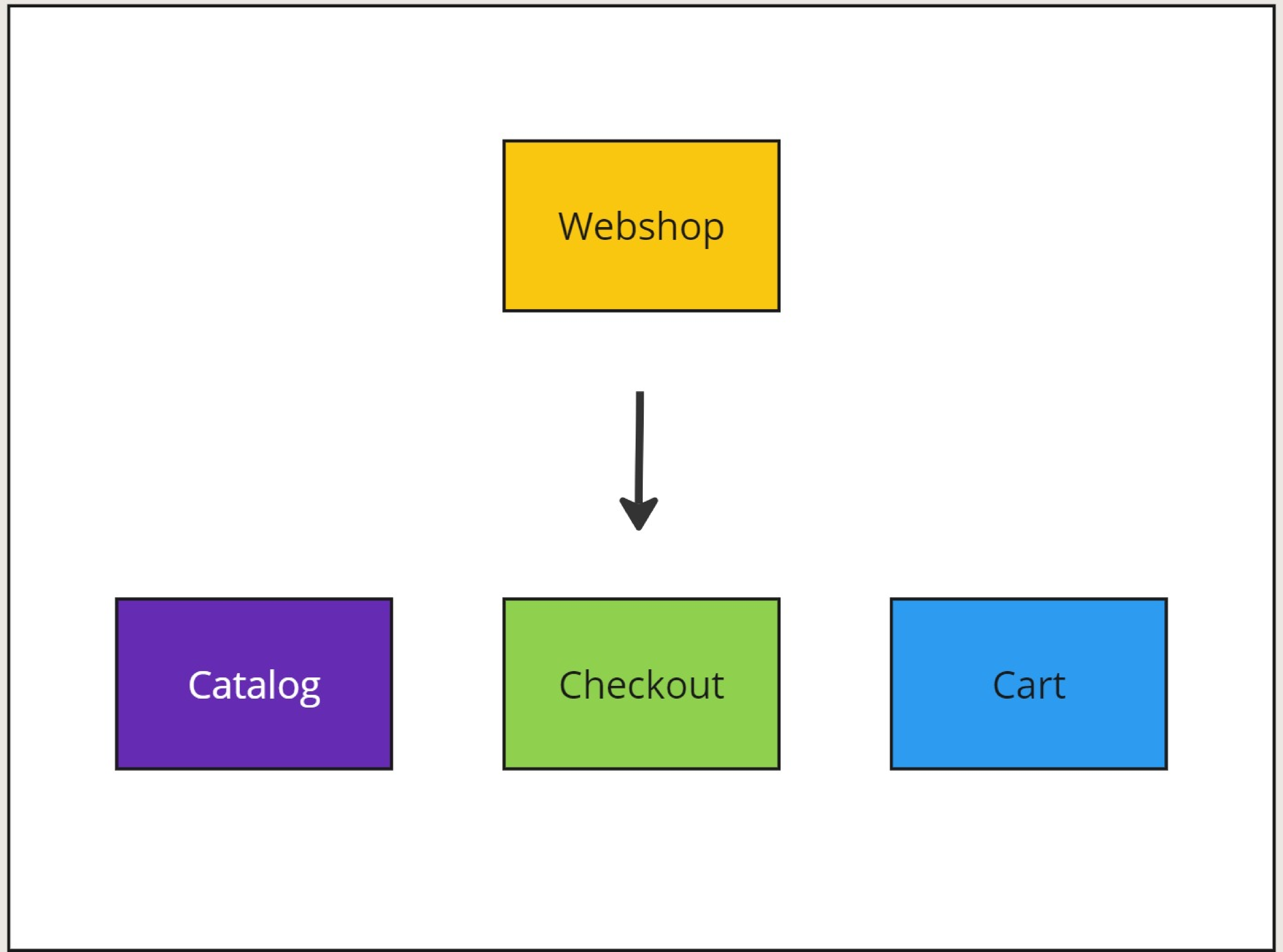
What can we do along the way?

When the purpose changes the architecture might need to grow.

You should always adhere to the following principles.

These principles are abstract and have no limits.

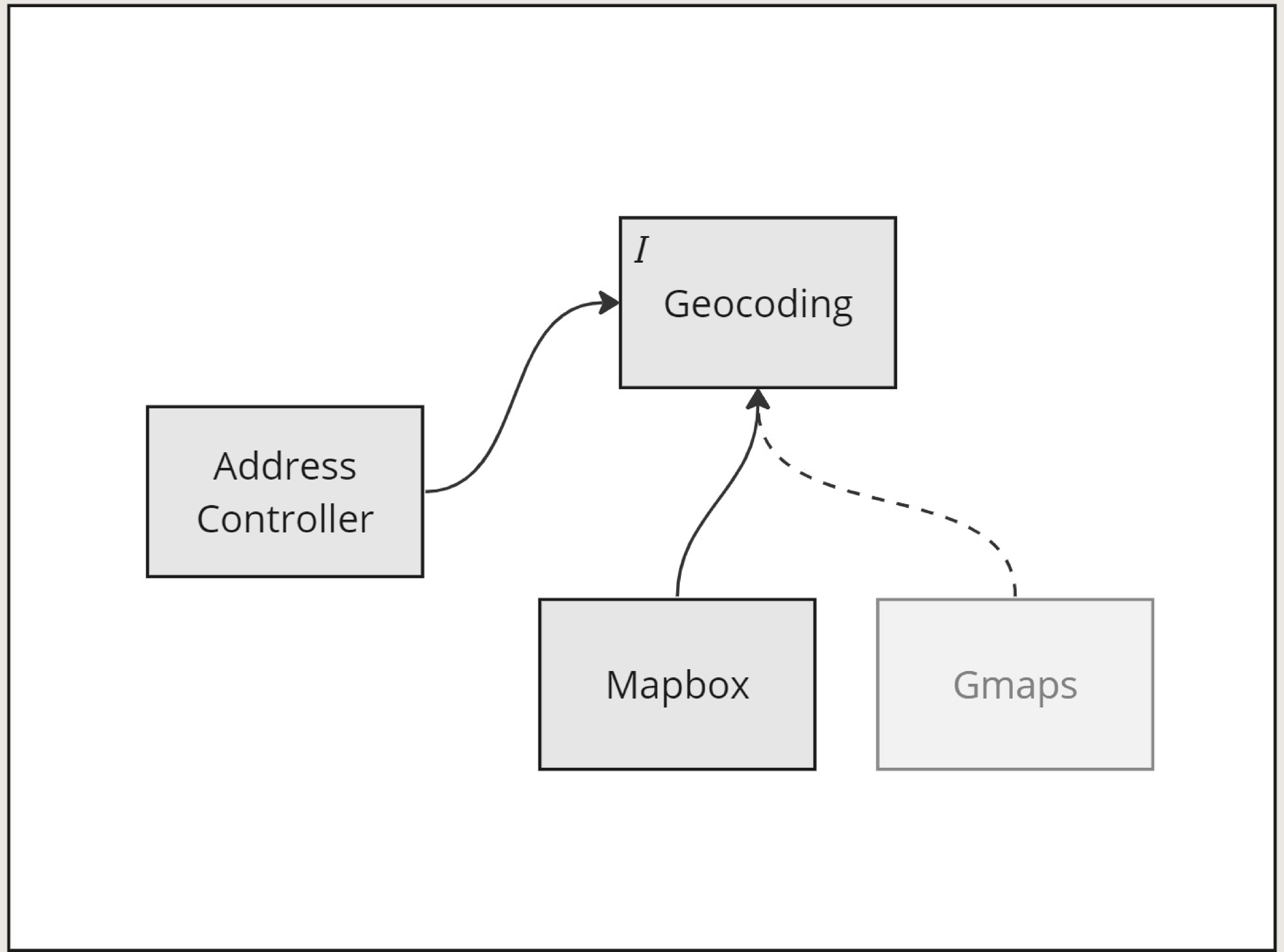
Single Responsibility Principle



A module should have only one reason to change.

It should do things in the scope of a single subject.

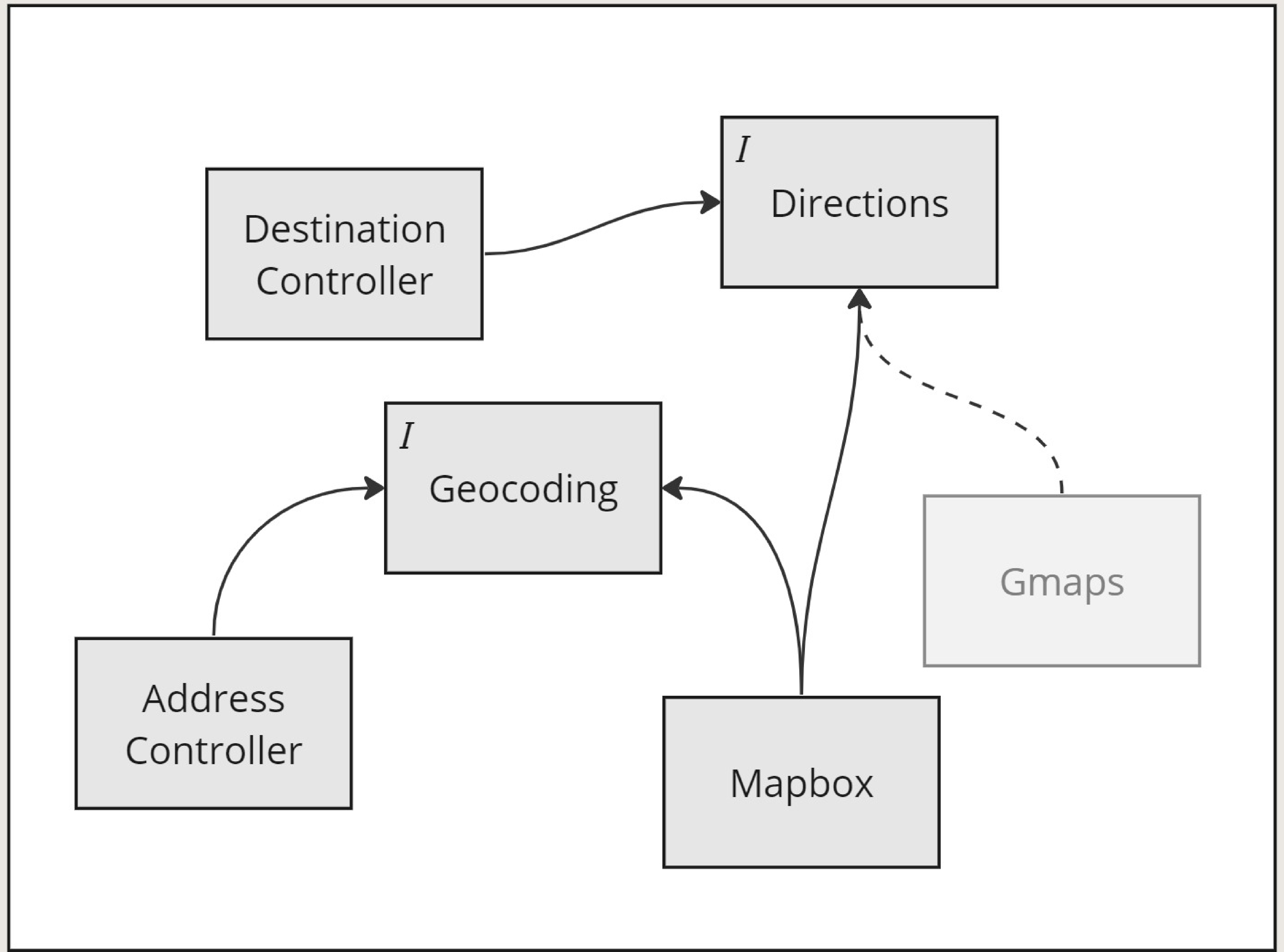
Dependency Inversion Principle



High-level modules should not depend on low-level modules.

It will decouple the low-level implementation from the high-level components.

Interface Segregation Principle



Clients should not be forced to depend on interfaces they do not use.

If the SRP and DIP had a child.

Applying Principles

All the principles

- Single Responsibility Principle
- Dependency Inversion Principle
- Interface Segregation Principle

Keeping options open

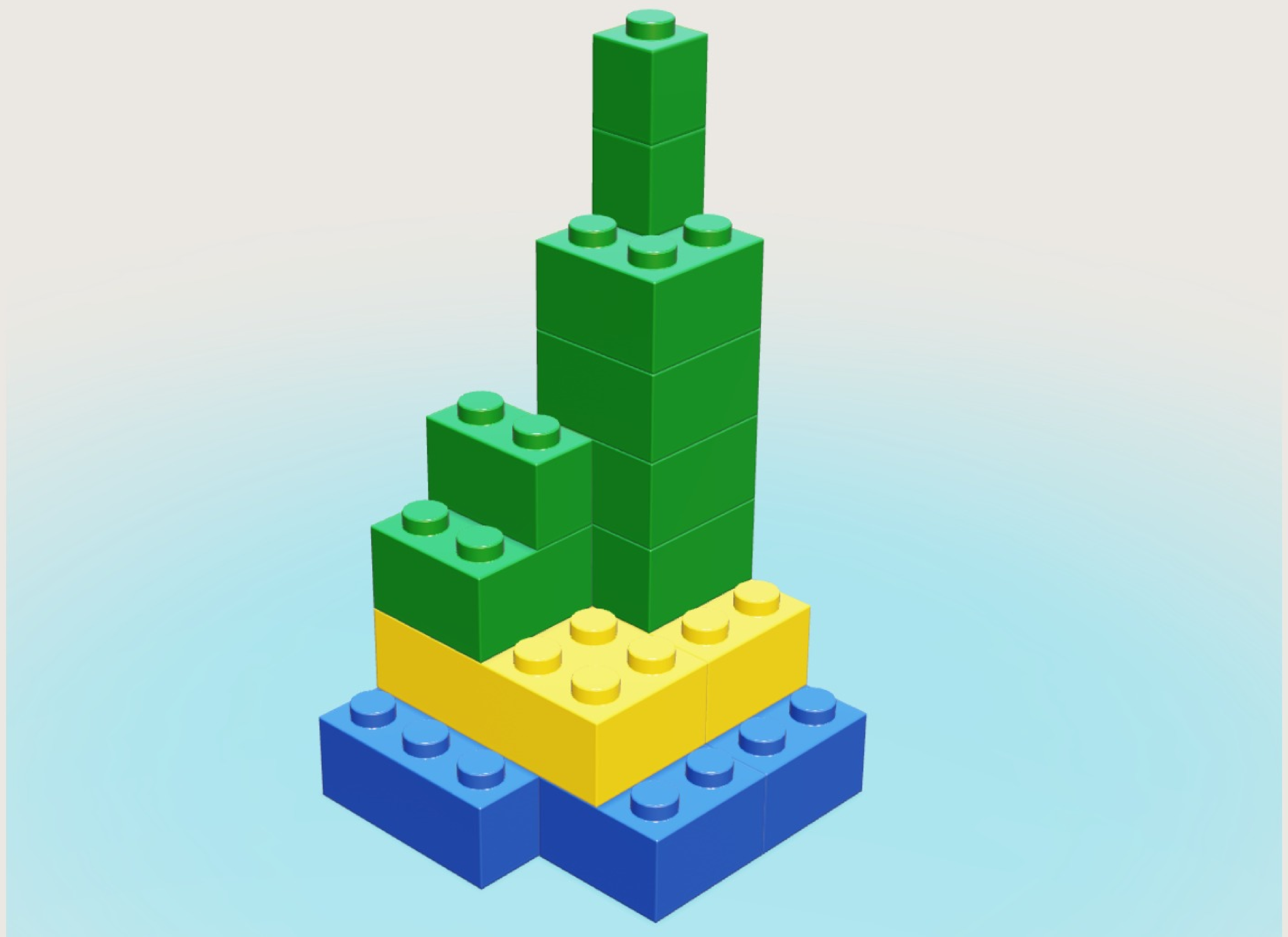
Separating the system into components with a single responsibility.

Isolating components through interfaces.

We open the pathways for the future.

Architecture

The monolith

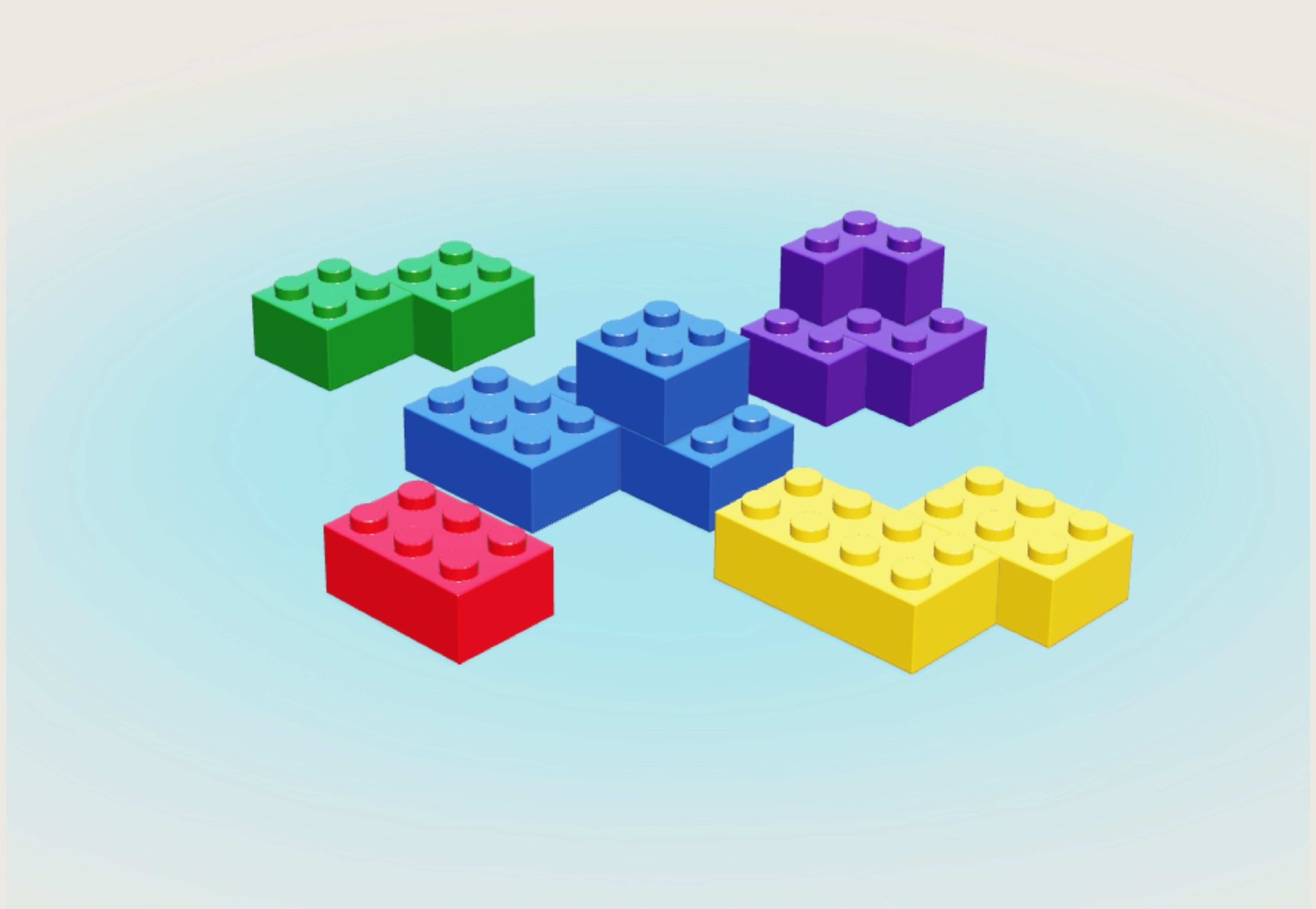


A monolith is a fine architecture for small applications.

It will get bigger when the applications grows.

Then it will become the skyscraper with all it's problems, the monolith.

Composable



Composable is decoupling the application.

We'll separate modules in to their own applications. Keeping their dependencies loosely coupled.

It won't be a monolith.

Umbraco & Architecture

When choosing an architecture it must support

- use cases of the system
- maintenance
- development
- deployment

Implementation fase

We need to code.
We need to commit.
We need to *blame*.
We need to accept.

Applying Umbraco

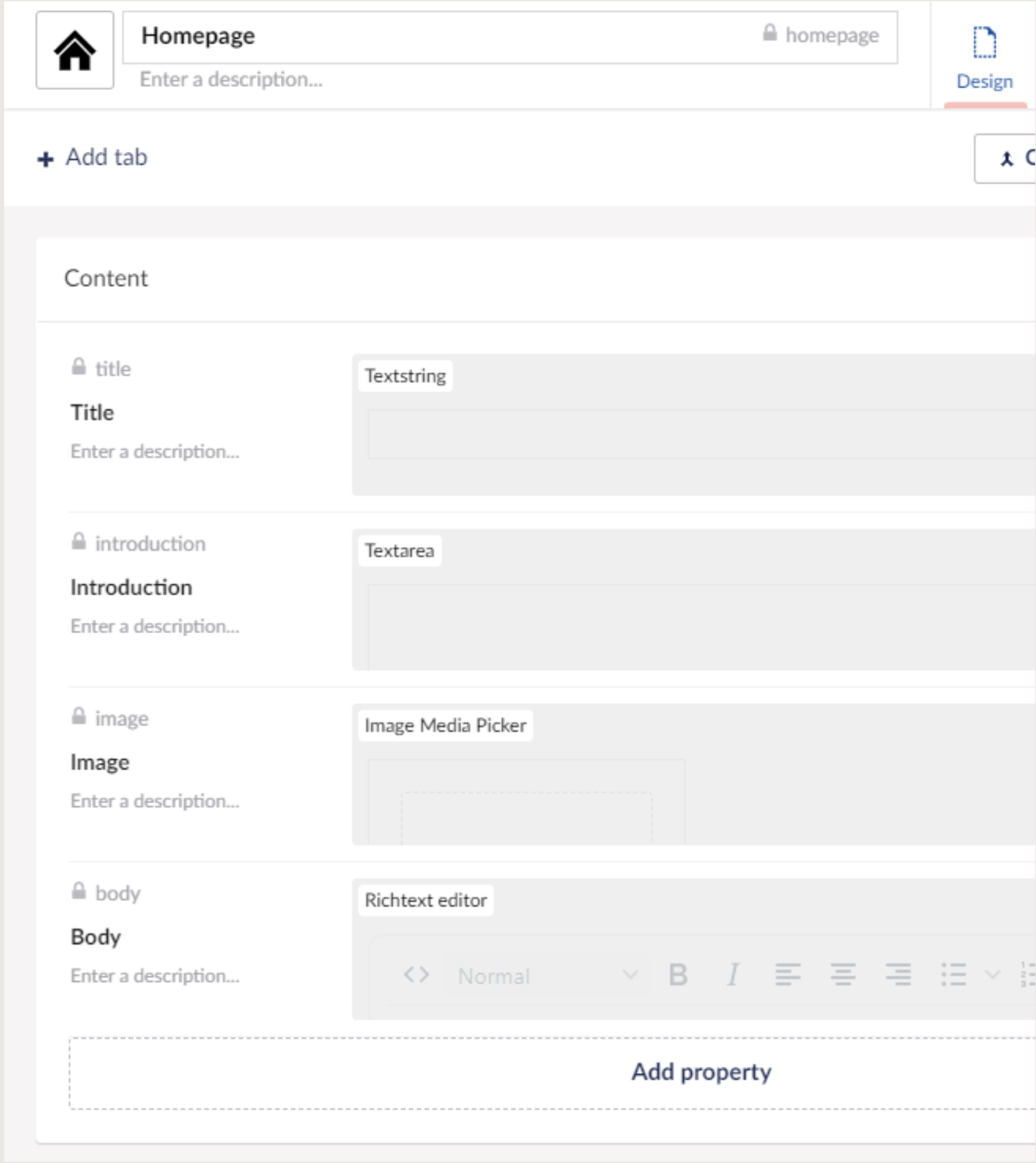
Let's dive in to Umbraco

How can we adhere to the principles when working with Umbraco?

Defining a document

We are defining a homepage.

I see two concerns here, we should separate it.



Homepage homepage
Enter a description...
Design
Add tab

Content

title
Title
Enter a description...
Textstring

introduction
Introduction
Enter a description...
Textarea

image
Image
Enter a description...
Image Media Picker

body
Body
Enter a description...
Richtext editor

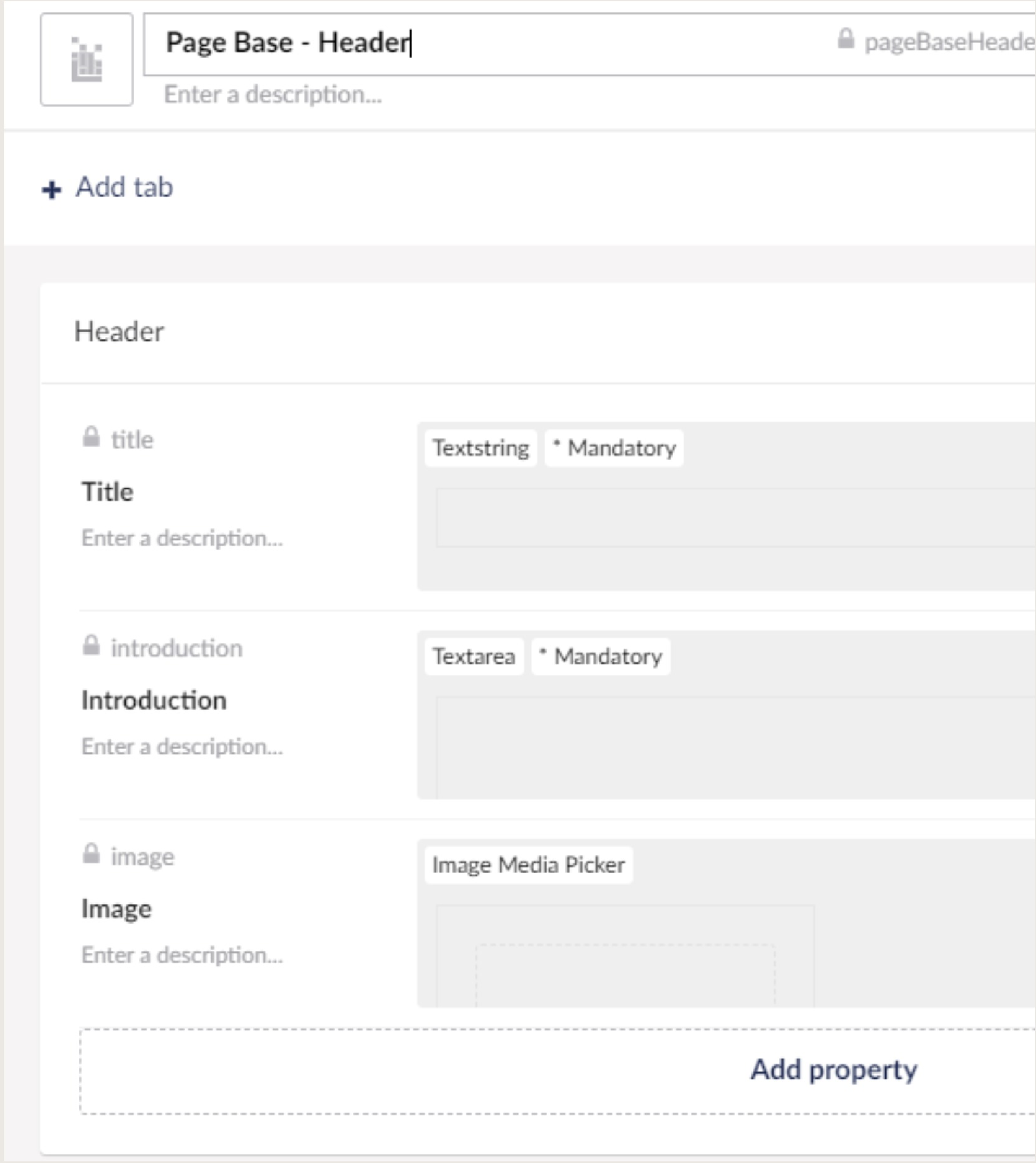
Add property

Defining a document

A composition is an interface.

We can reuse it.

It follows the single responsibility principle.



Page Base - Header

pageBaseHeade

Enter a description...

Header

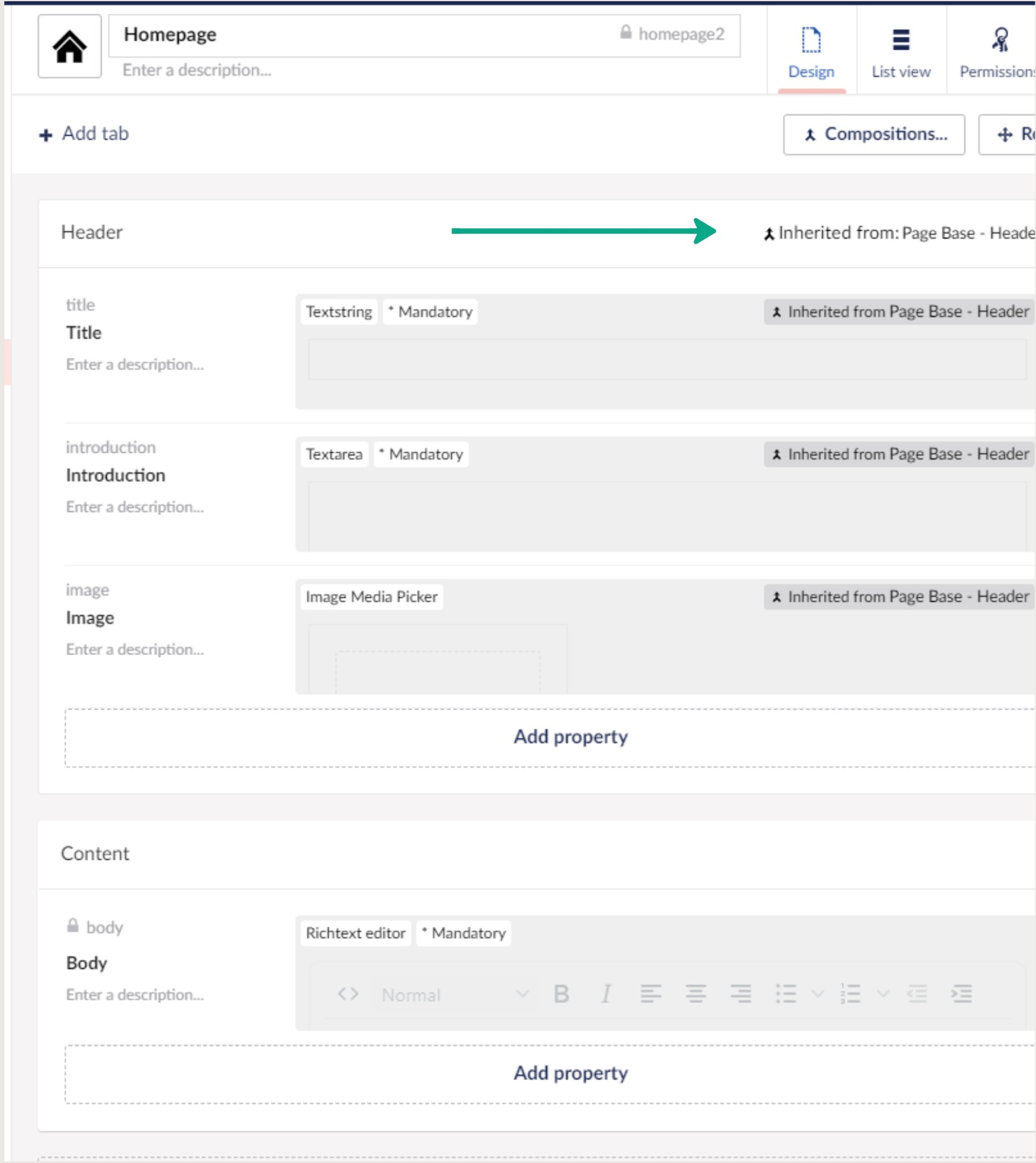
title **Title** Enter a description...	Textstring * Mandatory
introduction **Introduction** Enter a description...	Textarea * Mandatory
image **Image** Enter a description...	Image Media Picker

Defining a document

We can apply our composition to multiple document types.

The composition will create an interface in the Models Builder.

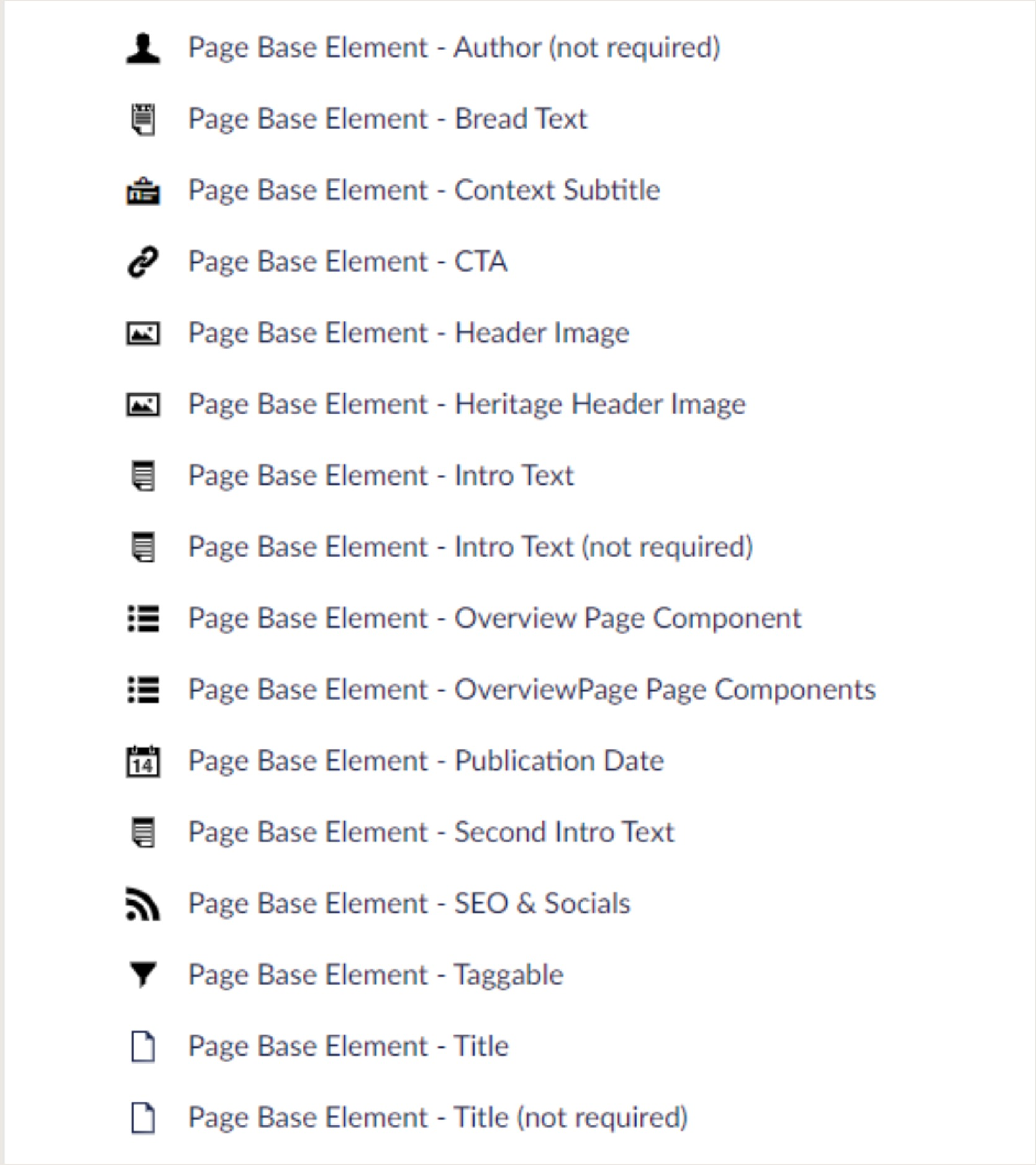
This will allow our code to follow the dependency inversion principle.



Defining a document

Create a composition for every element.

Follow the interface segregation principle.



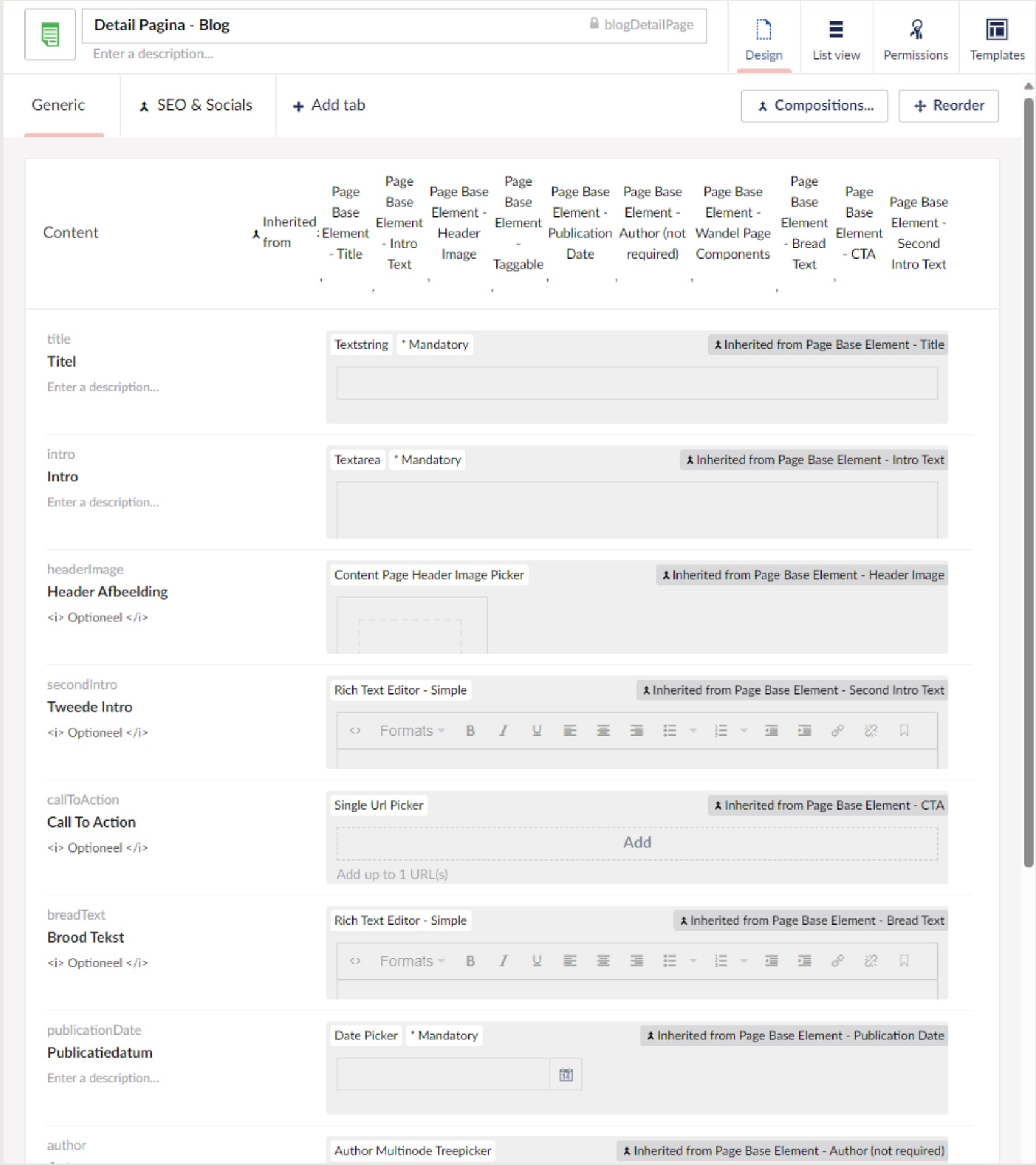
- Page Base Element - Author (not required)
- Page Base Element - Bread Text
- Page Base Element - Context Subtitle
- Page Base Element - CTA
- Page Base Element - Header Image
- Page Base Element - Heritage Header Image
- Page Base Element - Intro Text
- Page Base Element - Intro Text (not required)
- Page Base Element - Overview Page Component
- Page Base Element - OverviewPage Page Components
- Page Base Element - Publication Date
- Page Base Element - Second Intro Text
- Page Base Element - SEO & Socials
- Page Base Element - Taggable
- Page Base Element - Title
- Page Base Element - Title (not required)

Defining a document

We've split up the components and isolated them with interfaces.

Partial views will be written against the compositions interface.

Keeping our options open to changes.

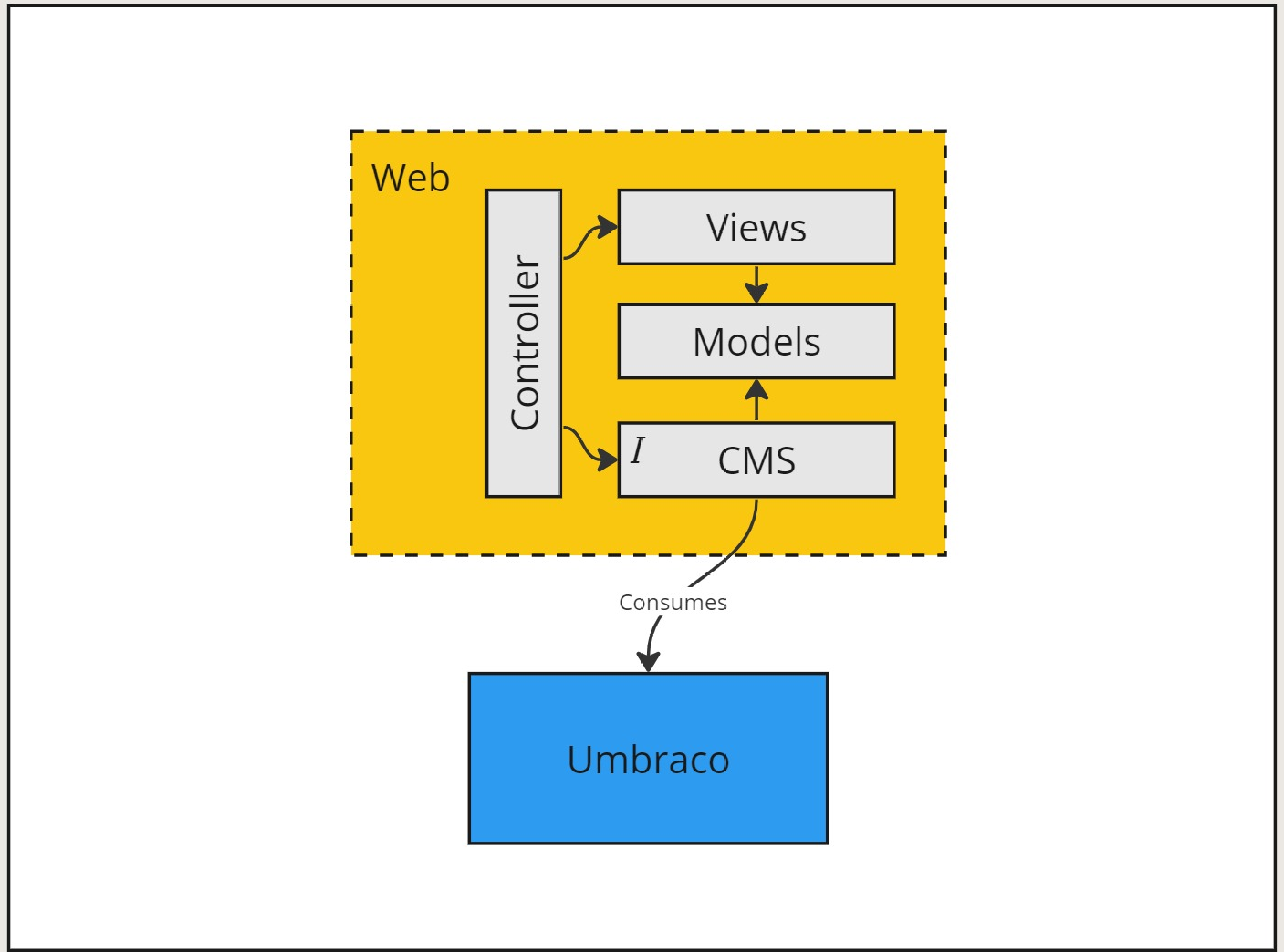

Detail Pagina - Blog
blogDetailPage
Enter a description...

Generic | SEO & Socials | Add tab | Compositions... | Reorder

Content
Inherited from: Page Base Element - Title, Page Base Element - Intro Text, Page Base Element - Header Image, Page Base Element - Taggable, Page Base Element - Publication Date, Page Base Element - Author (not required), Page Base Element - Wandel Page Components, Page Base Element - Bread Text, Page Base Element - CTA, Page Base Element - Second Intro Text

title **Titel** Enter a description...	Textstring * Mandatory	Inherited from Page Base Element - Title
intro **Intro** Enter a description...	Textarea * Mandatory	Inherited from Page Base Element - Intro Text
headerImage **Header Afbeelding** <i> Optioneel </i>	Content Page Header Image Picker	Inherited from Page Base Element - Header Image
secondIntro **Tweede Intro** <i> Optioneel </i>	Rich Text Editor - Simple	Inherited from Page Base Element - Second Intro Text
callToAction **Call To Action** <i> Optioneel </i>	Single Url Picker Add Add up to 1 URL(s)	Inherited from Page Base Element - CTA
breadText **Brood Tekst** <i> Optioneel </i>	Rich Text Editor - Simple	Inherited from Page Base Element - Bread Text
publicationDate **Publicatiedatum** Enter a description...	Date Picker * Mandatory	Inherited from Page Base Element - Publication Date
author	Author Multinode Treepicker	Inherited from Page Base Element - Author (not required)

Decouple Umbraco

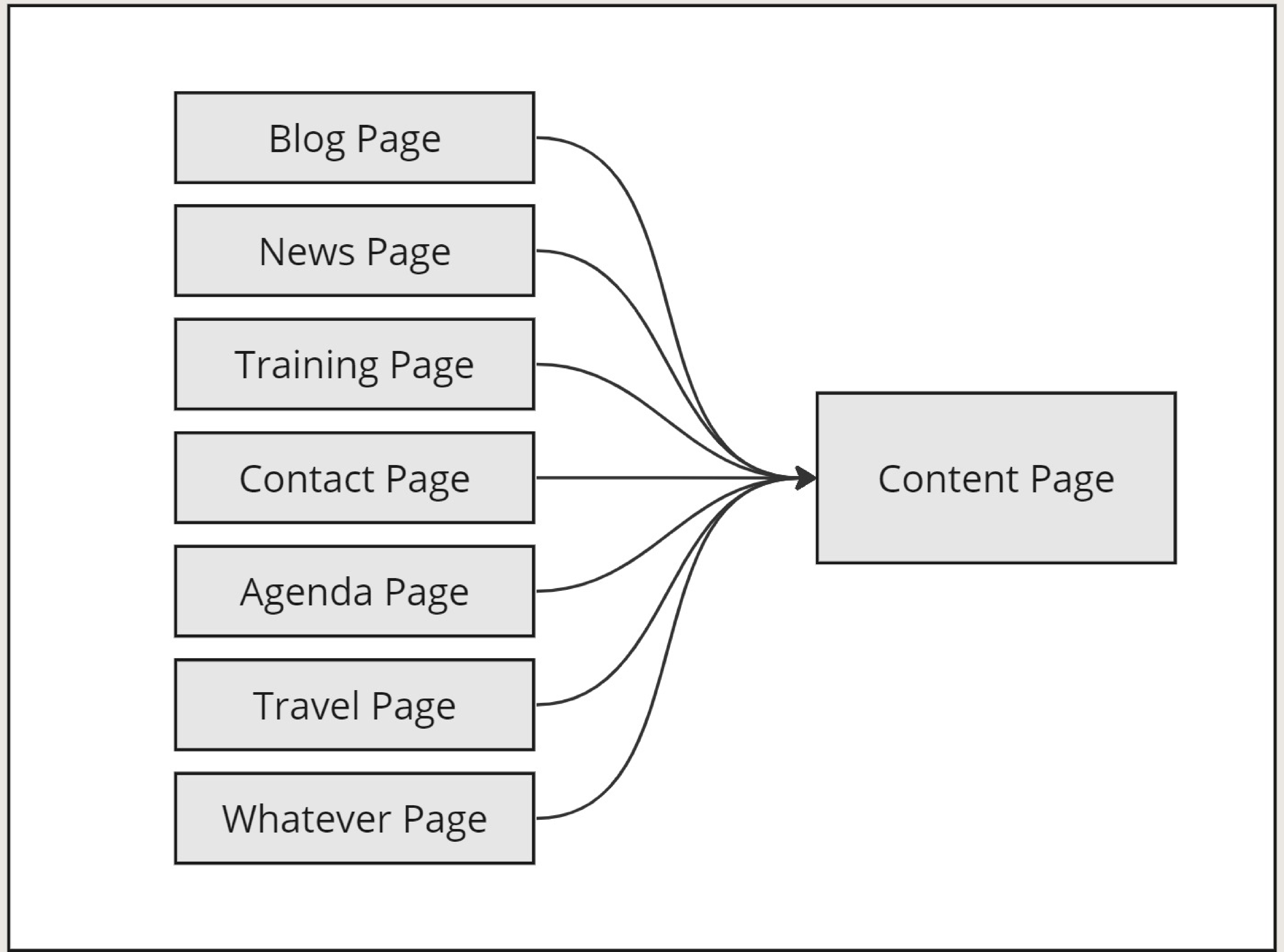


The website is not build on top of Umbraco.

It is build on top of a contract. Which the CMS implements.

Normalizing the different aspects of the website.

Composing pages

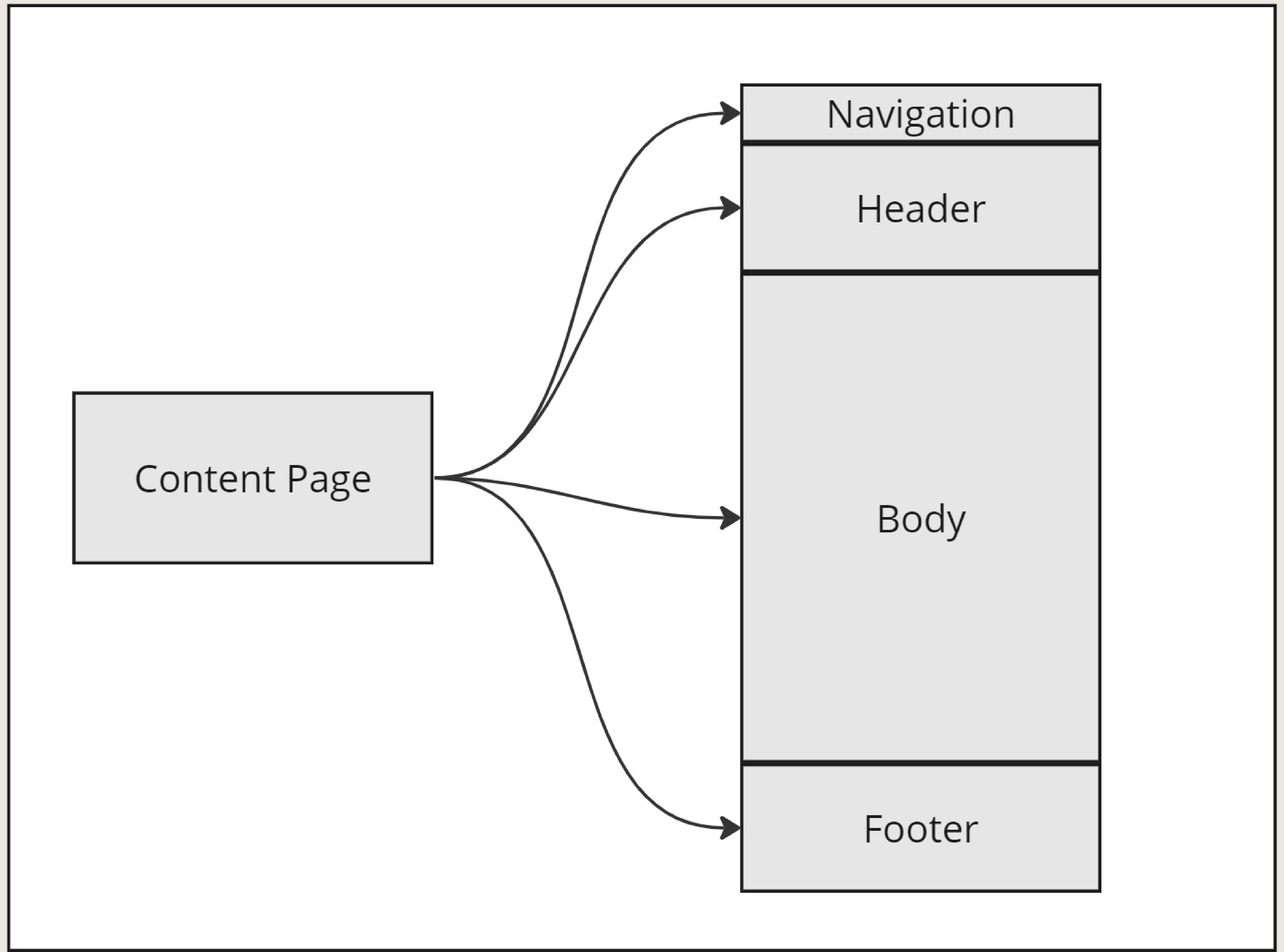


From a designers perspective there is only a content page template.

For the content editors there are many more.

The document types provide context. But is visually the same.

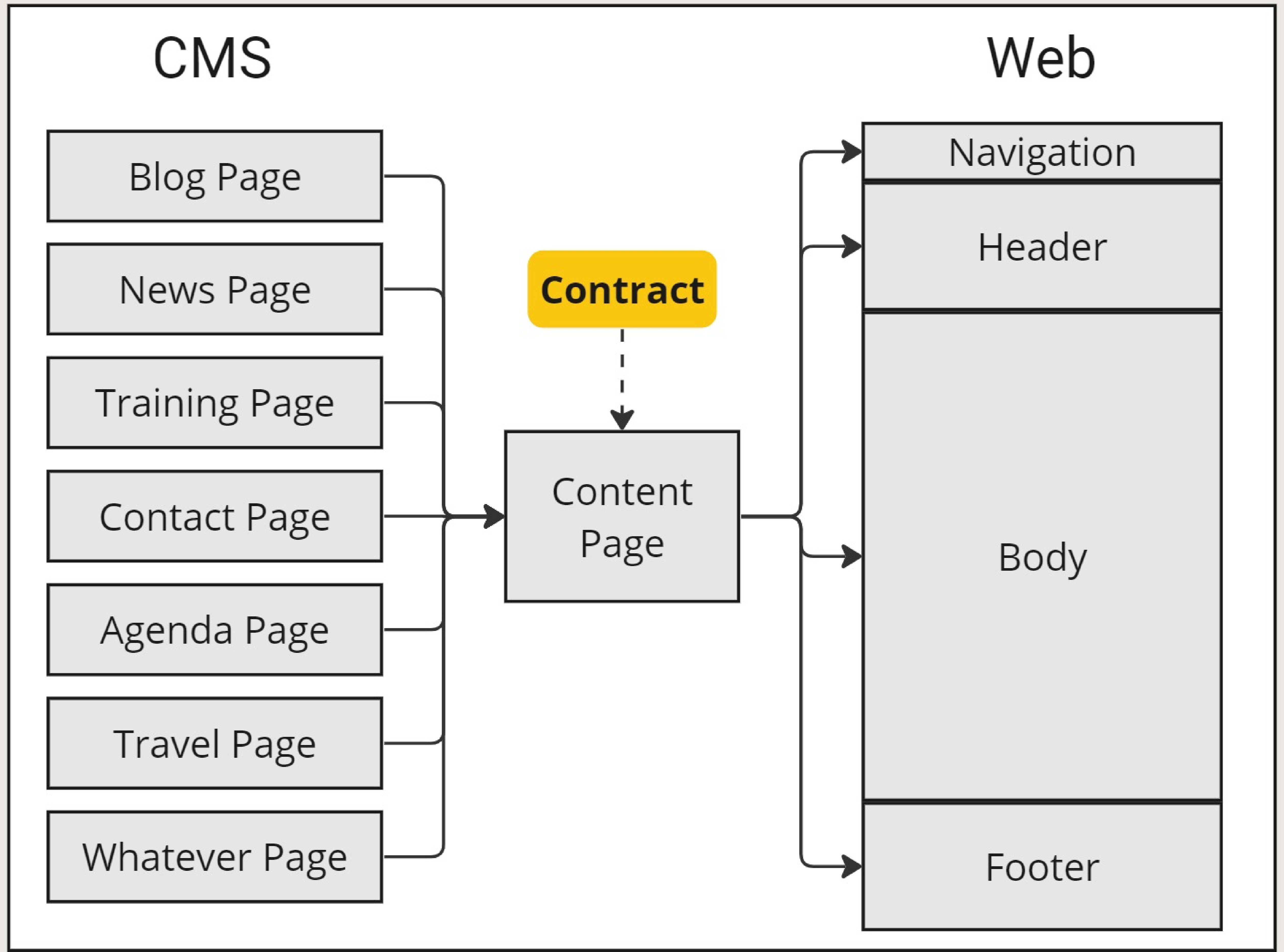
Composing the website



We can normalize these templates in to components.

And each component will have more layers to it.

Composing the website



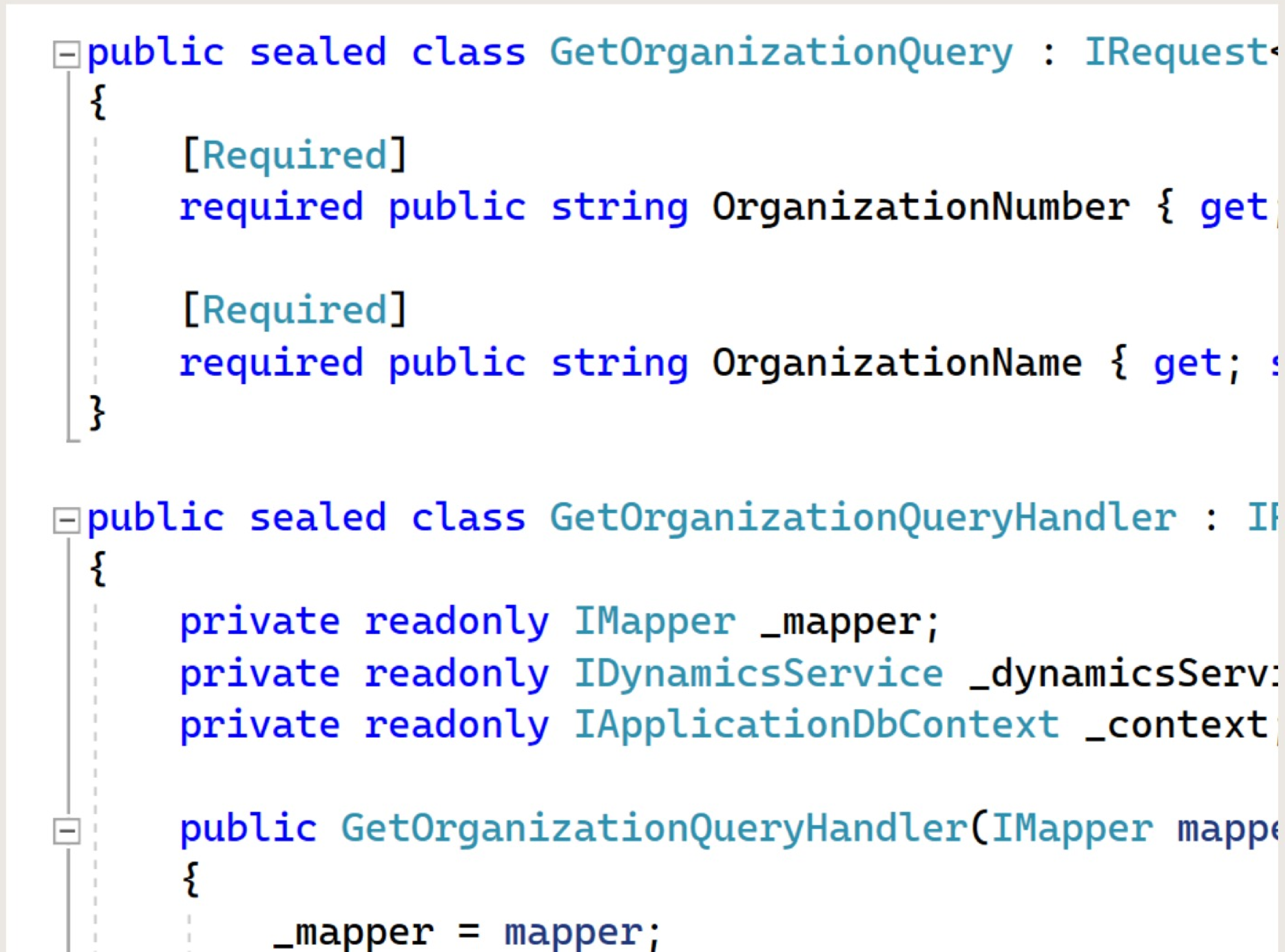
Everything is based on the contract of the content page.

There only needs to be one model for it.

And one view. Which has multiple components.

Code Maintenance

Code

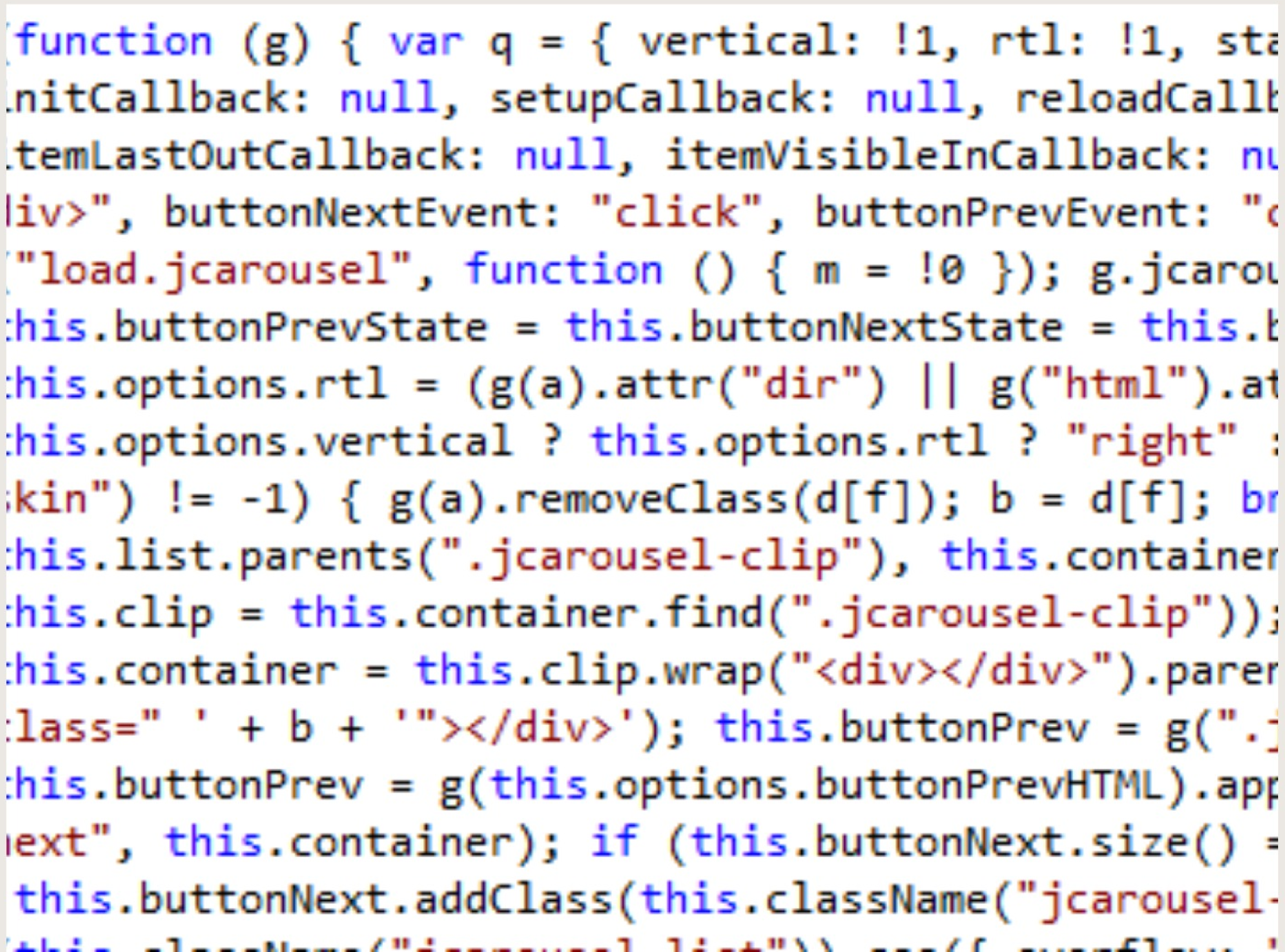


```
public sealed class GetOrganizationQuery : IRequest<
{
    [Required]
    required public string OrganizationNumber { get;

    [Required]
    required public string OrganizationName { get; s
}

public sealed class GetOrganizationQueryHandler : IR
{
    private readonly IMapper _mapper;
    private readonly IDynamicsService _dynamicsServi
    private readonly IApplicationDbContext _context;

    public GetOrganizationQueryHandler(IMapper mappe
    {
        _mapper = mapper;
```

A strong foundation on only the module level is not enough

We also need to keep our code maintainable.

Codestyle



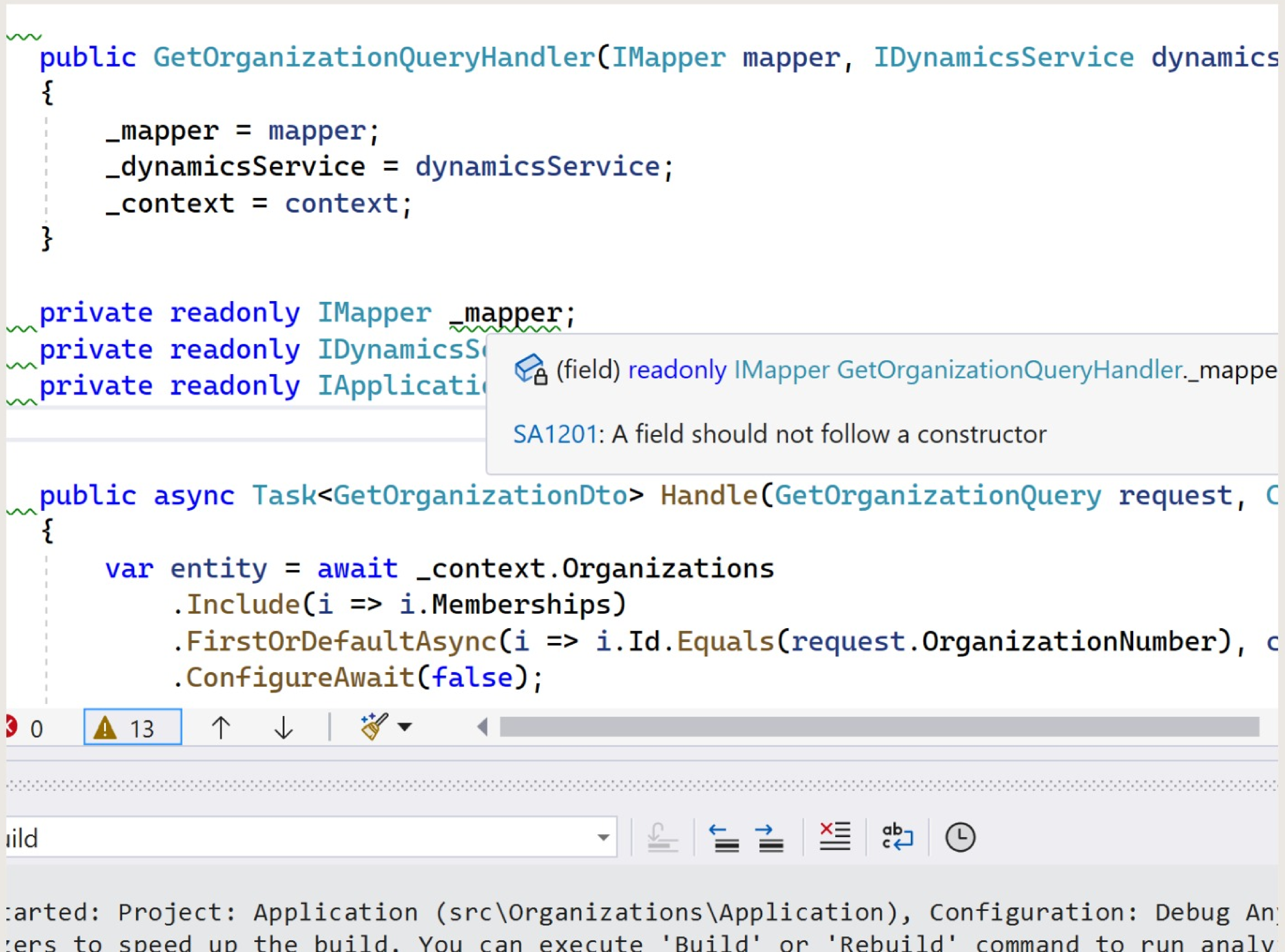
```
(function (g) { var q = { vertical: !1, rtl: !1, sta
nitCallback: null, setupCallback: null, reloadCallb
temLastOutCallback: null, itemVisibleInCallback: nu
iv>", buttonNextEvent: "click", buttonPrevEvent: "c
("load.jcarousel", function () { m = !0 }); g.jcarou
his.buttonPrevState = this.buttonNextState = this.b
his.options.rtl = (g(a).attr("dir") || g("html").at
his.options.vertical ? this.options.rtl ? "right" :
kin") != -1) { g(a).removeClass(d[f]); b = d[f]; br
his.list.parents(".jcarousel-clip"), this.container
his.clip = this.container.find(".jcarousel-clip"));
his.container = this.clip.wrap("<div></div>").paren
lass=" ' + b + '"></div>'); this.buttonPrev = g(".j
his.buttonPrev = g(this.options.buttonPrevHTML).app
ext", this.container); if (this.buttonNext.size() =
 this.buttonNext.addClass(this.className("jcarousel-
this.className("jcarousel-list")).css({ overflow: '
```

Solve it by having and maintaining a code style.

The team should code as one.

Avoid technical dept.

Gate keeping



```
public GetOrganizationQueryHandler(IMapper mapper, IDynamicsService dynamics
{
    _mapper = mapper;
    _dynamicsService = dynamicsService;
    _context = context;
}

private readonly IMapper _mapper;
private readonly IDynamicsS
private readonly IApplicati

public async Task<GetOrganizationDto> Handle(GetOrganizationQuery request, C
{
    var entity = await _context.Organizations
        .Include(i => i.Memberships)
        .FirstOrDefaultAsync(i => i.Id.Equals(request.OrganizationNumber), c
        .ConfigureAwait(false);
```

(field) readonly IMapper GetOrganizationQueryHandler._mappe

SA1201: A field should not follow a constructor

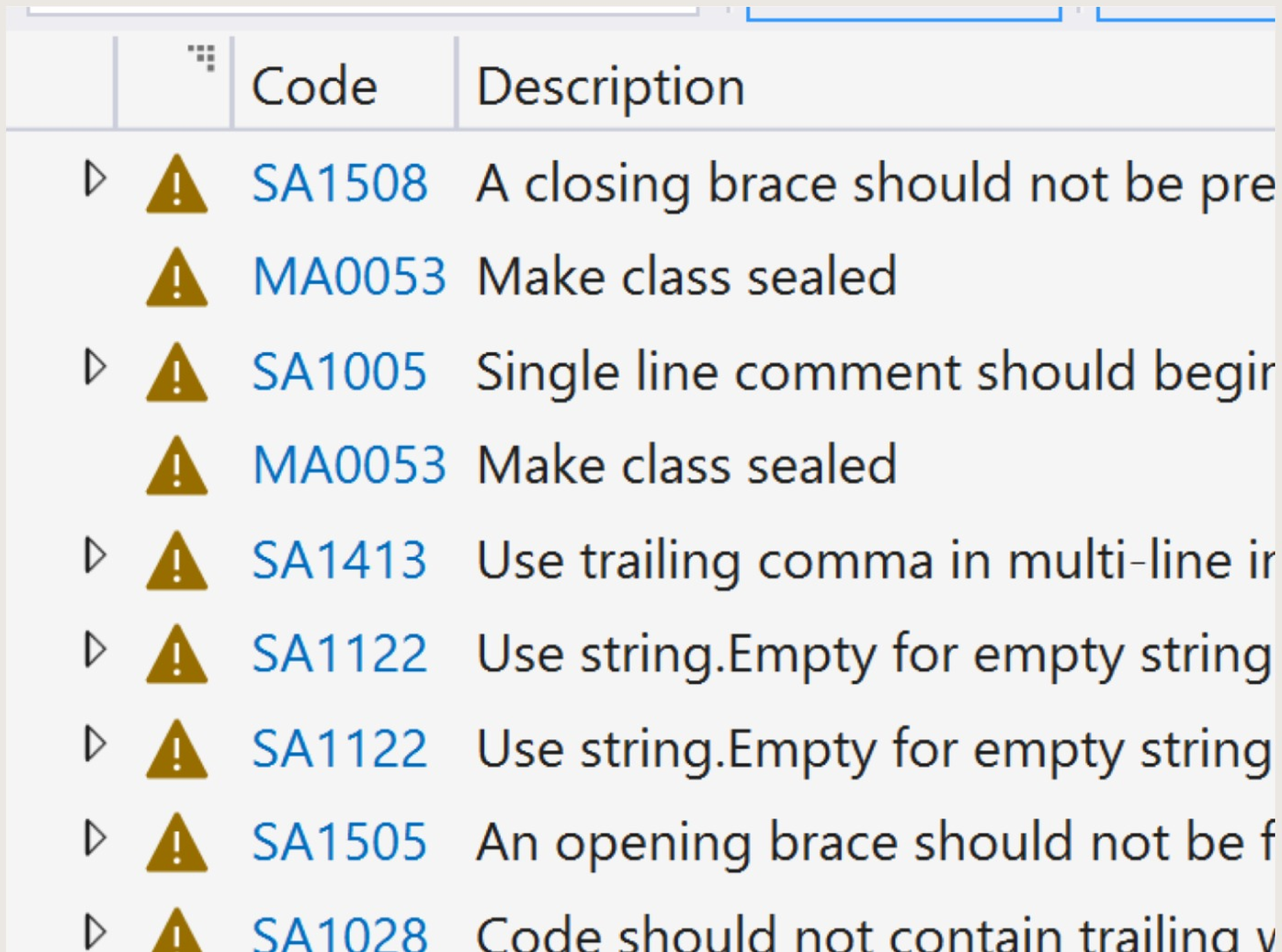
```
tarted: Project: Application (src\Organizations\Application), Configuration: Debug An
zers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analy
```

Defining a codestyle is one. Applyin it is a second.

Use static code analysis to warn you about codestyle offenses.

Use tools such as StyleCop and SonarQube.

Warnings == Errors



	Code	Description
⚠	SA1508	A closing brace should not be pre
⚠	MA0053	Make class sealed
⚠	SA1005	Single line comment should begir
⚠	MA0053	Make class sealed
⚠	SA1413	Use trailing comma in multi-line ir
⚠	SA1122	Use string.Empty for empty string
⚠	SA1122	Use string.Empty for empty string
⚠	SA1505	An opening brace should not be f
⚠	SA1028	Code should not contain trailing v

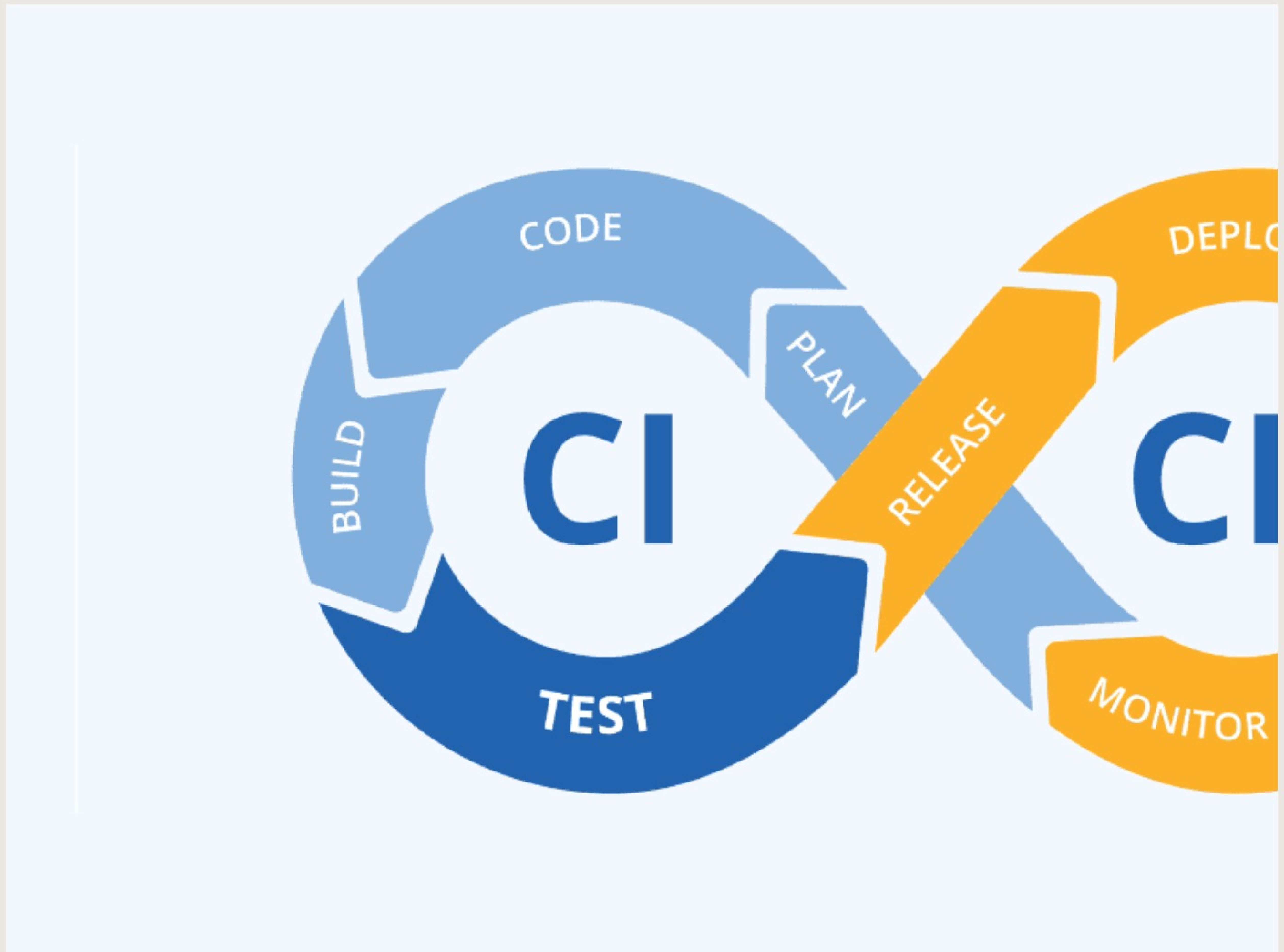
Warnings are errors.

A warning is something that is not breaking to the function of the application.

It is breaking to the maintainability of the code.

CI/CD

Continuous Integration

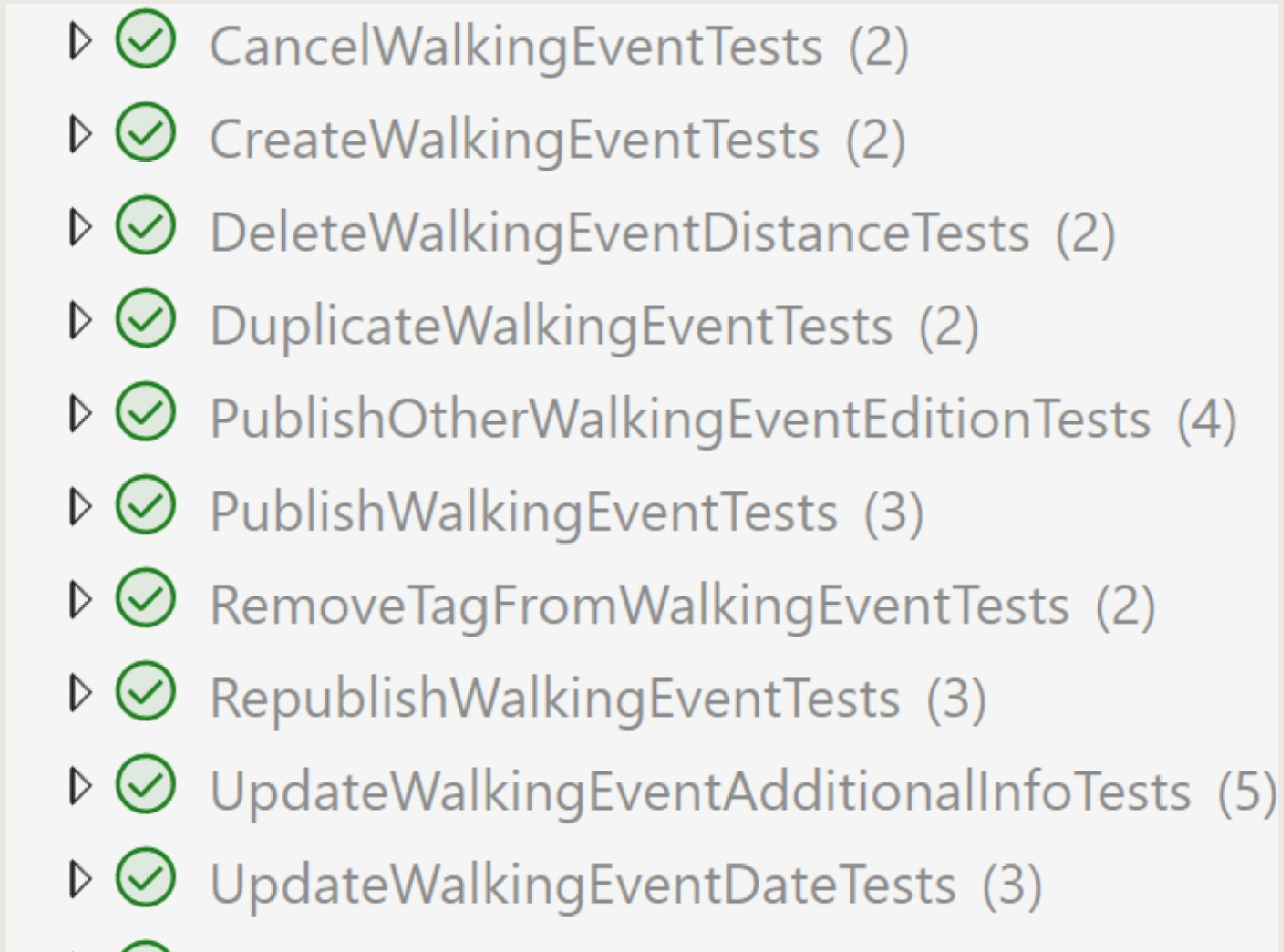


Continuously integration our code back in to a shared repository.

Testing our code automatically pre-merge.

Keeping bugs at a distance.

Automated Testing



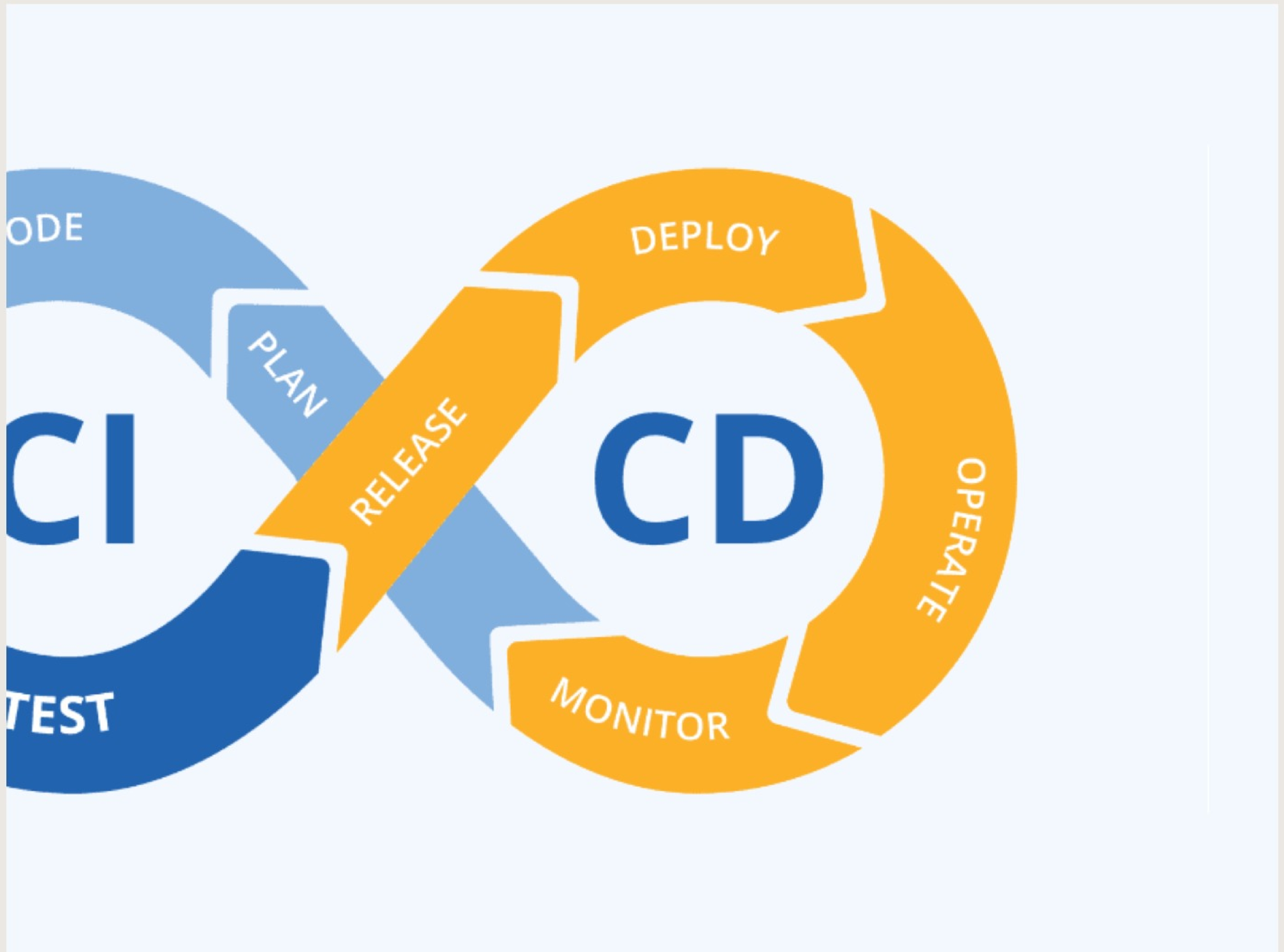
CancelWalkingEventTests (2)
CreateWalkingEventTests (2)
DeleteWalkingEventDistanceTests (2)
DuplicateWalkingEventTests (2)
PublishOtherWalkingEventEditionTests (4)
PublishWalkingEventTests (3)
RemoveTagFromWalkingEventTests (2)
RepublishWalkingEventTests (3)
UpdateWalkingEventAdditionalInfoTests (5)
UpdateWalkingEventDateTests (3)

Testing exists in many forms. In units or as a whole integration.

Require tests to succeed in the DevOps process (pre-merge).

Require codestyle to be resolved (pre-merge).

Continuous Deployments

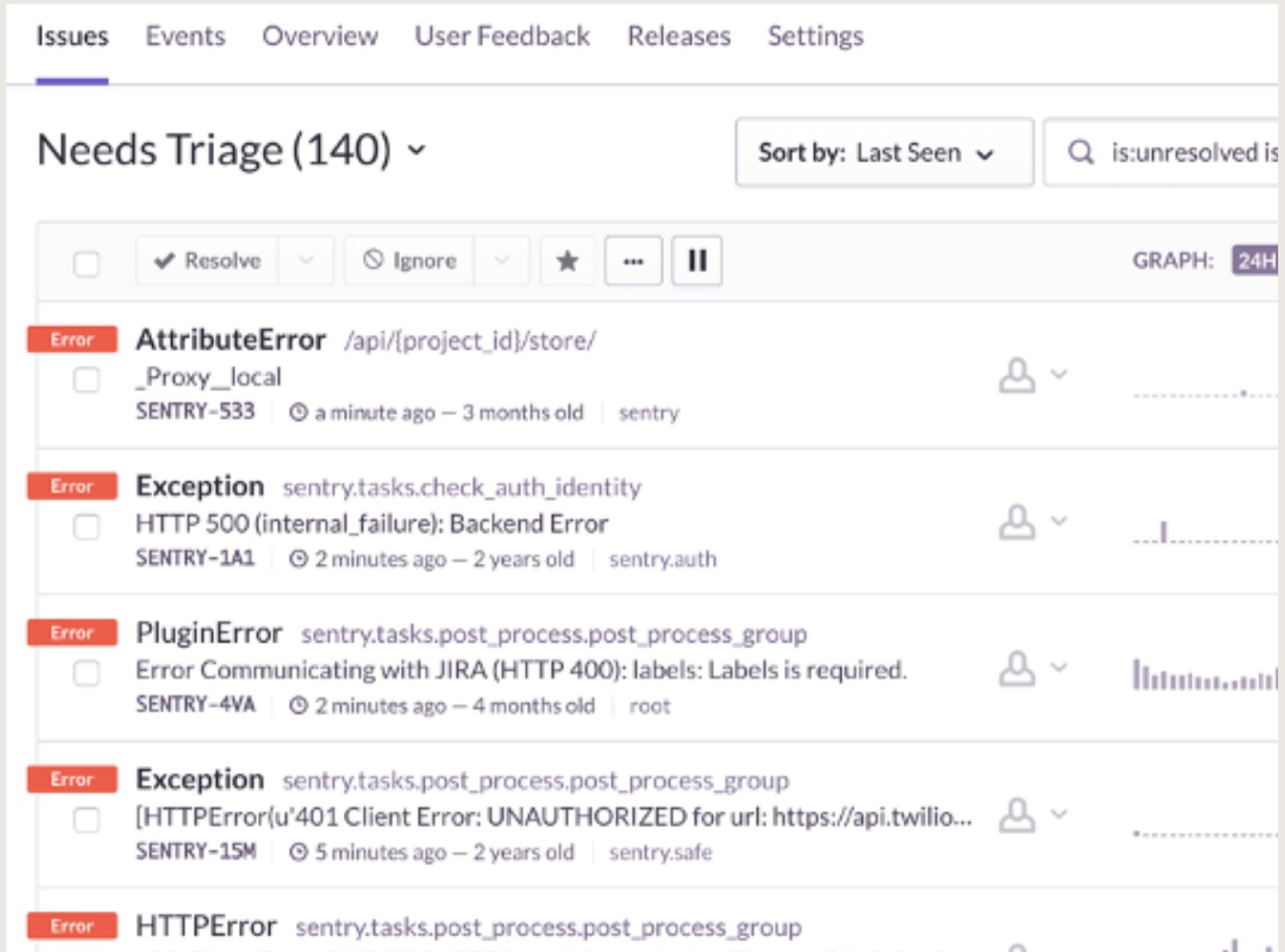


This doesn't only mean an 'automated deploy'.

It means monitoring and error management.

Use tools such as Sentry or New Relic.

Monitoring



Error AttributeError /api/{project_id}/store/
_Proxy__local
SENTRY-533 a minute ago — 3 months old sentry

Error Exception sentry.tasks.check_auth_identity
HTTP 500 (internal_failure): Backend Error
SENTRY-1A1 2 minutes ago — 2 years old sentry.auth

Error PluginError sentry.tasks.post_process.post_process_group
Error Communicating with JIRA (HTTP 400): labels: Labels is required.
SENTRY-4VA 2 minutes ago — 4 months old root

Error Exception sentry.tasks.post_process.post_process_group
[HTTPError(u'401 Client Error: UNAUTHORIZED for url: https://api.twilio...
SENTRY-15M 5 minutes ago — 2 years old sentry.safe

Error HTTPError sentry.tasks.post_process.post_process_group

Apply performance monitoring and error tracking in your application.

Find bottlenecks or broken code.

Plan to fix optimalizations or bugs before it is too late.

That's all

Questions?

Roy Berris

@royberris

www.berris.dev