



Masterthesis Computer Science

Building a Website with a Static Site Generator

Phil Elgert

30.09.2022

Examiner

Prof. Dr. Torsten Grust

Co-Examiner

Prof. Dr. Thomas Walter

Supervisor

Denis Hirn, Tim Fischer

Phil Elgert:

Building a Website with a Static Site Generator

Masterthesis Computer Science

Eberhard Karls Universität

From 01.04.2022 to 30.09.2022

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterthesis selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterthesis wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum

Phil Elgert

Abstract

Static websites are really fast but oftentimes time-consuming in maintenance because of their lack of code abstraction. In this work, Static Site Generators (SSGs) are examined. These are frameworks that make the implementation and maintenance of a static website really easy while still preserving the speed benefit of static websites. A real world website is implemented using the open-source SSG Hugo. Core concepts are explained, using examples from the website. Furthermore a documentation of the implemented parts and components is given. In addition, a fast deployment pipeline that is based on Git is provided.

Contents

Abstract	v
Acronyms	ix
1 Introduction	1
2 Objective	3
2.1 Content	3
2.2 University Corporate Design	3
2.3 Maintainability	3
2.4 Responsiveness	3
2.5 Git Support and Deployment	4
2.6 Static Website	4
3 Deployment Pipeline	5
4 Harald? - No, my name is Hugo	7
4.1 Content Files - Pages	7
4.2 Templates	8
4.3 Go Templates	10
4.4 Partials and Shortcodes	11
4.5 Page Bundles	11
4.6 Themes	11
4.7 Archetypes	12
4.8 Site Configuration	12
5 Implementation	15
5.1 Base Layout	15
5.1.1 Menu	17
5.1.2 Section Menu	18
5.2 Content Structure	19
5.3 Section Components	20
5.3.1 Team	21
5.3.2 Teaching	24
5.3.3 Publications	27
5.3.4 Research	29
5.3.5 Theses	30
5.3.6 News	31
5.4 Additional Components	32
5.5 Build Environments	34
5.6 JS/CSS compilation	34
5.7 Archetypes	35
6 Conclusion	37
Bibliography	39

Acronyms

CMS Content Management System

CSS Cascading Style Sheets

HTML Hyper Text Markup Language

IDE Integrated Development Environment

JS JavaScript

RSS Really Simple Syndication

SASS Syntactically Awesome Style Sheets

SSG Static Site Generator

UI User Interface

URL Uniform Resource Locator

YAML YAML Ain't Markup Language

Introduction

Nowadays, each group or organization that is large enough has a website. There are around 200 million active website as of 2022 [1]. This is because it is a fast, easy and cheap way to supply information. Websites can be categorized in a lot of ways, the most prominent being *static* and *dynamic*. A static website basically has no user interaction. It just presents information in the form of text, images, etc. There is no login and there is no live search. A static website is like a museum: have a look but do not touch! On the other hand there is dynamic websites. On these websites there is usually a user login and there is a lot of interaction with the user. A user can perhaps change the state of the website, by editing content that is for example saved in a database. Since dynamic websites have all these functions, they are oftentimes slower and are high-maintenance. Therefore, if this kind of functionality is not needed, running a static website is the way to go.

In this thesis, the domain of SSGs is explored. These are frameworks that make content management and implementation of a static website more convenient by adding for example code abstraction, front matter and control flow structures. Many more concepts will be explained in this thesis. The topic is explored using a practical example. The website of the Database Systems chair [2] of the University of Tübingen was reimplemented. The old website had an outdated design and needed to be updated using the new cooperate design of the University. Using examples from the website, the most important concepts of SSGs are explained. Additionally, a really fast deployment pipeline for the website is presented. The pipeline works with Git [3], which is the most important versioning control system on the market.

This thesis also serves as the documentation of the website. The most important components and their usage will be explained.

Objective

The goal of this thesis is to lay out how to implement a website using a SSG. The concrete implementation was done for the Database Systems Research Group at the University of Tübingen.

To make the objective clearer, a handful of conditions are set for a successful implementation.

2.1 Content

Since there already existed a website of the chair, the content had to be integrated into the new website. The old website used Trello. Trello is a Content Management System (CMS) where content is defined in Markdown [4] files. This is convenient because in Hugo, content is defined using Markdown as well. Therefore most of the content is reused for the new website. Static files, like images and PDFs are declared in Hugo syntax.

2.2 University Corporate Design

The former website of the chair was using an outdated design that leaned on the former corporate design of the University. To have the website up-to-date with the design of the University [5], the components and styles of the website are taken from it.

2.3 Maintainability

To successfully run a website over many years, it has to be easy to maintain. For this reason, it is important for the website to have the code split up into reusable components.

2.4 Responsiveness

As most website visitors use mobile devices, a website is required to be responsive. This means that on every device, be it small or large, the website has to be rendered in a way

the user can easily use, read and interact with all its content properly. Hence, on smaller devices an open menu typically uses up all the space on the screen to make sure the user is able to target the menu items with his finger. On big screens it might be rendered as a box next to the navigation bar which is opened on hover or on click. It does not use up the entire screen but only a part of it.

The responsiveness was already implemented in the Cascading Style Sheets (CSS) of the design, which are reused.

2.5 Git Support and Deployment

The code of the website needs to be accessed and changed by multiple people, which should not result in chaos and an inconsistent website being served. This means that code maintenance and deployment has to be easy. Therefore the goal is to use Git [3] to have versioning of the source code. Additionally, a deployment pipeline is needed to rebuild and serve the website upon code changes.

2.6 Static Website

To use an SSG for a site, the site has to be static. This means that between releases, the website stays the same. In contrast, dynamic sites often have a database backend. The content of the database, which is eventually displayed on the page, can change at any time. Thus the content of the website changes in spite of it not being rebuilt.

Deployment Pipeline

In this chapter the deployment process of the website is explained.

The code is versioned with Git. In this way anyone that has access to the Git repository of the website can make changes to it. The changes get deployed immediately. In addition, Git provides a commit history of the changes made to the website so that one can always check which person made which changes at what time. Another advantage of using Git and a deployment pipeline is that the website can be edited via command line. A content manager does not need to use a User Interface (UI) to make changes to the site.

In Figure 3.1 there is the structure of the pipeline which is responsible for deploying this website.

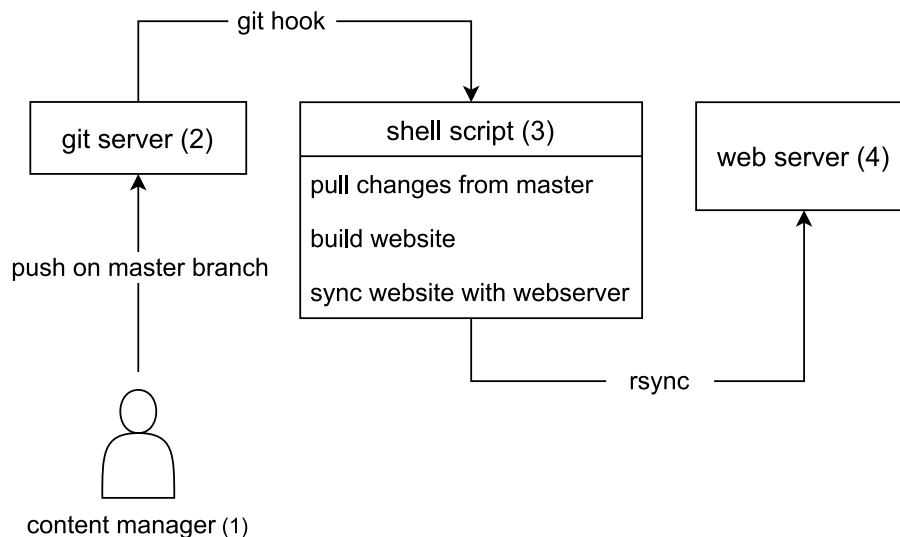


Figure 3.1: Deployment Pipeline

The pipeline works as follows:

1. Whenever a content manager makes changes to the website that need to be deployed, he pushes them on the master branch of the website repository.
2. The Git server receives all changes for each of its repositories. When it receives a change on the master branch of the website repository, a Git hook script is run.
3. The hook script pulls new commits from the master branch of the Git repository.

Then, the website is built with Hugo. Hugo outputs a static website. The website is synced with the website of the web server in the root directory. This is done with the command line tool **rsync** which synchronizes all files with the files on the web server file-by-file. In so doing the website does not experience any downtime. In contrast, just deleting and copying the new website on the web server would keep the website in an inconsistent state for just a short time. That is because during the process there are always some missing files, or in the worst case all of them (when deleting is done and copying about to start).

4. The web server runs during the whole process. It gets served upon receiving the new version of the website.

Harald? - No, my name is Hugo

In order to create a website and host it on a server, the most basic way is to create a Hyper Text Markup Language (HTML) file for each page. This works, but it also has a huge downside since most of the pages contain a lot of the same content, for example, the header, the footer and the menu. Content that is repetitive should be refactored. This is where for example SSGs can come into play as they provide the ability to refactor code and reuse it in multiple pages.

For this thesis the SSG Hugo [6] is used. It provides these features and many more that are discussed in this chapter.



Figure 4.1: The Hugo logo

4.1 Content Files - Pages

Each Hugo content file represents one page of the website. In Hugo, content is typically written in Markdown (HTML is also possible, but not really used). Markdown brings the advantage of its simple and intuitive syntax. Simple content parts like paragraphs, images, captions, links (and many more) are supported by Markdown, so one does not have to bother with HTML tags.

A content file in Hugo contains two parts: the front matter and the content. In Figure 4.2, there is an example of a content file from the website. In the top part there is the front matter, enclosed between 3 dashes ---. Front matter is written in YAML (YAML Ain't Markup Language). After that, the content is defined which consists of some markdown and a call to a shortcode at the bottom (line 10).

Front matter is how one defines meta data for a content file. This meta data can be attributes which Hugo provides out of the box like the date of creation of the file, the title or Uniform Resource Locator (URL) aliases. But Hugo also allows the definition of custom front matter in content files.

Front matter cannot be directly accessed from a content file. But it can be accessed from

```

1 ---
2 title: Data Provenance for SQL
3 aliases:
4   - /research/data-provenance-for-sql
5 researchProject: data-provenance-for-sql
6 ---
7 We explore new ways to derive the provenance (or lineage) of data
8   items that flow through programs or queries. Once this provenance
9   information has been derived, we know
10
11 1. exactly which input items led the program (or query) to emit which
12   output items (Why and Where Provenance), as well as
13 2. which program parts were involved in the computation of each
14   single item (How Provenance).
15
16 {{< partial "publications/by-project" >}}

```

Figure 4.2: A content file in Hugo. Line 1-6 is front matter written in YAML. Line 7-9 is content written in Markdown. In line 10, there is a call to a shortcode.

templates, partials and shortcodes. One can think of partials and shortcodes as functions that produce an HTML output (see Section 4.4). In line ?? of Figure 4.2, there is the `researchProject` custom parameter set. This parameter is accessed and used by the shortcode that is called in line 10.

When the website is built with the `hugo` command, all Markdown is compiled into HTML. Each partial and shortcode is interpreted and the resulting HTML inserted into the content.

In Hugo, most pages are single pages. They display information about a single piece of data. Pages of the same type of data reside in the same folder, for example, all news articles are in the news folder and therefore under the `/news` URL. For each folder there is normally one list page. This page displays a summary of the information from all single pages in that folder. In case of the news list page, it displays a summary for each news article.

In Figure 4.3, there is the structure of the news section. List pages always have the file name `_index.md`. Single pages have either the name `index.md` if they have a page bundle (see Section 4.5). Or without page bundle, it can have any name that fits.

4.2 Templates

Every Hugo project has one base template which is the `baseof.html` template. It contains the HTML that is the same on each page. This is where the footer and the header are

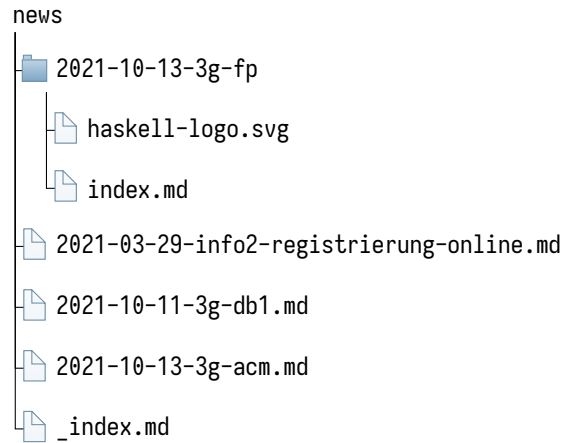


Figure 4.3: The folder structure of the news section. The `_index.md` file is the list page for the news. It renders a summary of all pages in the news folder. Every other Markdown file represents a single page.

```
1 {{ define "left-sidebar" }}
2   {{ if (not .Page.Params.contentFullWidth) }}
3     {{ partial "left-sidebar/left-sidebar.html" . }}
4   {{ end }}
5 {{ end }}
6 {{ define "main" }}
7   {{ .Content }}
8 {{ end }}
```

Figure 4.4: `list.html/single.html` - The list/single page template. At the top, the `left-sidebar` is rendered. The sidebar is only rendered if the page does not have the front matter parameter `contentFullWidth: true`. Underneath, the content of the page is rendered.

defined, because they appear on every page.

In addition to the base template, each content file is rendered using another more specific template which is the list or single page template. If there are parts of the page which are the same for every single page but not for list pages, one would put them into the single page template. There is also the option of making templates section-specific. One could for example add a single page template for the news section which would be used to render all news single pages. So for each page the most specific template and the `baseof.html` is used.

In this project there is a list and a single page template which are identical. In Figure 4.4, the template definition can be seen.

```

1 {{ $main := resources.Get "/css/page.css" }}
2 {{ $custom := resources.Get "/sass/custom.sass" | toCSS }}

3 {{ $css := slice $custom $main | resources.Concat "css/main.css" |
   minify | fingerprint }}

4 {{ return $css }}

```

Figure 4.5: `compileCss.html` - a returning partial, that does not produce any HTML, but returns a computation result. CSS is bundled with compiled SASS. Then it is minified and fingerprinted

4.3 Go Templates

In order to write templates in Hugo, one is in need of control structures, variables and functions. One needs for example the ability to loop through the pages of a section, or to recursively build the menu. To express this functionality, Hugo makes use of Go templates. These are control structures that are expressed in the Go language [7]. Go templates can be used in any template file, partial or shortcode. This template syntax is depicted by the double curly braces `{{ }}` that contain the commands. In Figure 4.4 there is an example. A context is passed to each layout template, partial or shortcode. This context contains the page that is being rendered with all its front matter and content as well as all site variables and functions. One can access this context in Go Templates. The context is depicted by `.` ("the dot"). Line 2 of Figure 4.4 illustrates the way the `contentFullWidth` custom parameter is accessed from the context of a page. Line 3 shows how the context is passed to the `left-sidebar.html` template. Line 8 depicts how the content of the page is taken from the context and rendered.

Hugo provides a lot out-of-the-box functions that can be used in templates. Figure 4.5 displays template that compiles, merges, minifies and fingerprints all CSS and Syntactically Awesome Style Sheets (SASS). Figure 4.6 shows the definition of the `styles.html` partial. It is called in the header of the page. It compiles the CSS with the `compileCss.html` partial and includes it into the website with an HTML link tag.

```

1 {{ $pageCss := partial "func/compileCss" . }}
2 <link rel="stylesheet" type="text/css" href="{{ $pageCss.Permalink }}"
  >

```

Figure 4.6: `styles.html` - Styles are compiled and included into the site with a link tag

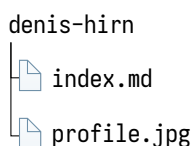
4.4 Partials and Shortcodes

In programming there is a paradigm called the DRY (do not repeat yourself) paradigm. It means that redundancy in code is to be avoided. It is bad practice for example to put the footer of the website into each content file. One would rather refactor it into a partial and reference the partial instead. Hugo supports this partial functionality in templates. One can pass parameters and the context to a partial. The context (aka "The dot" `.`) contains all site variables, config parameters and the current page from which the partial is called. From this context, front matter can be accessed.

For content files, the same functionality is provided in shortcodes. To use partials in both templates and content files, the project defines a shortcode named `partial.html`. This shortcode takes a name of a partial to be called and passes the context down to the partial. There is an example in Figure 4.2 in line 10.

4.5 Page Bundles

Sometimes a page consists of multiple files. For example the person's profile can contain a picture of the person or a publication usually has a PDF version for download. To add these to the context of the page, they need to be in the same directory as the page and declared in the front matter of the page. In Figure 4.7 there is an example page bundle for a team member. Since the resources also exist in the context of each page, they can be accessed from any template. In Section 5.3.1 there are example usages of the `portrait` page resource.



```
denis-hirn
├── index.md
└── profile.jpg
```



```
1 resources:
2   - name: portrait
3     src: profile.jpg
4 ---
5 {{< person/single >}}
```

Figure 4.7: On the left there is the directory structure of the page bundle for `/team/members/denis-hirn`. On the right there is the resource definition in the content file (`denis-hirn/index.md`). The `person/single` shortcode uses this page resource to render the profile image.

4.6 Themes

Hugo provides the concept of themes to separate content and layout. There are a lot of community supported themes at <https://themes.gohugo.io/>. These themes can be im-

ported as a Git submodule into a project. This is a good choice if there is no time to produce one's own custom layout or someone just wants to create a simple homepage.

For this project, the out-of-the-box themes were only used as inspiration on how to structure the layout. The University of Tübingen has its own cooperate design which had to be implemented as a Hugo theme. With regard to implementation, this means that all layout components, such as partials, shortcodes, the base template, and the CSS lie in the themes directory. In order to register a theme in the project, it has to be declared in the config file of the site.

4.7 Archetypes

New content files can be created just by adding new files manually to the content folder. The boilerplate code can be copied from other pages and used as guidance. When working with a graphical editor or an Integrated Development Environment (IDE) this is often the easiest way.

But Hugo provides a command-line alternative, called archetypes. An archetype reflects the structure that each page of a given type has in common. In Figure 4.8 for instance, on the left, one can see all the code structure that is found in each of the member pages.

Creating pages with the help of archetypes is done via the **hugo** command line tool. The syntax is the following: **hugo new --kind <kind> <path/to/destination>**. Currently supported **<kind>**s are: member, alumni, lecture, publication, thesis, news. The file name that is passed as the destination path (last bit of the destination URL) is passed to the archetype template as the parameter **Name**.

When creating new content through an archetype, the whole page bundle of the archetype is copied to the destination. In addition, the Go Templates (not the shortcodes!) in the archetype definition are evaluated.

Archetypes reside in the archetypes folder of the root directory.

4.8 Site Configuration

Hugo provides a wide range of different site configurations. These are defined in the config files that are found in the **config** folder. Each config file lives in a separate folder. The folders define different environments, like production or development. Depending on what command is run to build the site, a different config file is used. There is also the **_default** folder whose config file serves as a default config that gets merged with the other configs. Running the Hugo development server with file watch is done via the **hugo server** command. In this case the development config is used and merged with the default config. When the website is built using the **hugo** command, the production configuration is used

merged with the default config.

There is a default config and a development config in this project. The default config defines the `baseUrl` (the domain) of the site, the theme that is used and some other things. The development config contains just one entry: the `baseUrl` as `'/'`. The reason is that in production, the website will be hosted on a public domain, which has to be declared in the config file. But for development, the site should just be deployed to localhost. See Section 5.5 for more concrete information.

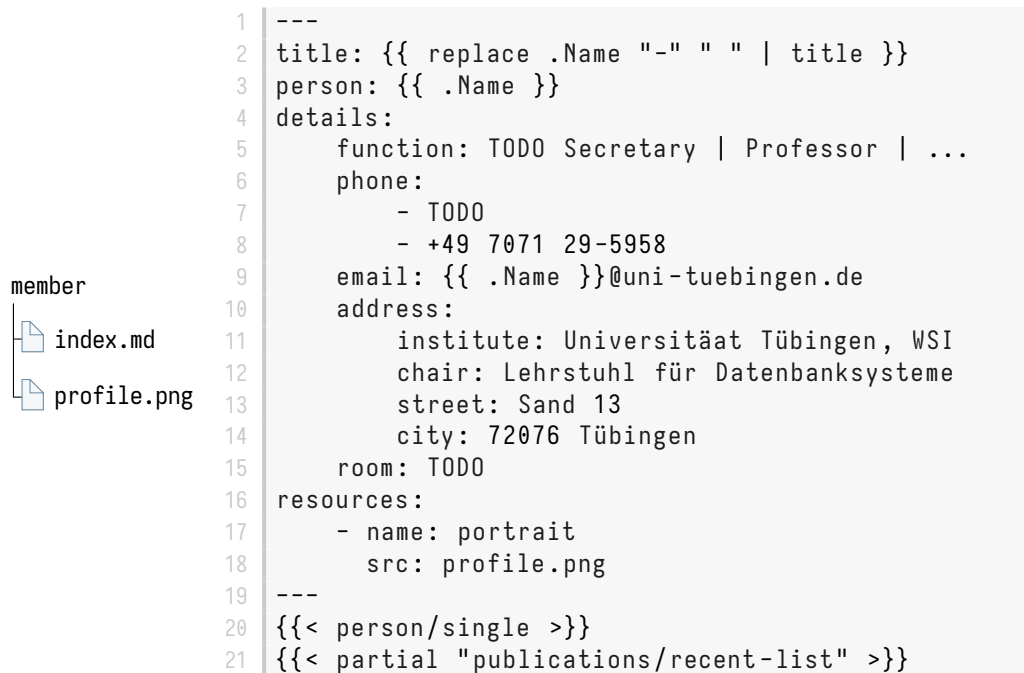


Figure 4.8: On the left there is the structure of the `member` archetype which is copied. Team members come with a dummy profile picture (`profile.png`). On the right there is the definition of the archetype. The Go templates in line 2, 3 and 9 are evaluated at creation time: The name that is passed to the archetype forms the title, person key and email. This name is the file name.

Implementation

Now that the foundation for basic Hugo concepts has been laid, the specifics about the actual implementation of the website will be given.

The structure of the website and its components will be presented in this chapter.

5.1 Base Layout

In Hugo, there is a template called the `baseof.html` template. It is the base template that defines the outer most structure of the web page.



Figure 5.1: The home page on a mobile device. The menu is moved to the top of the screen and opens on click. The `right-aside.html` block is moved to the bottom of the page (not visible here).

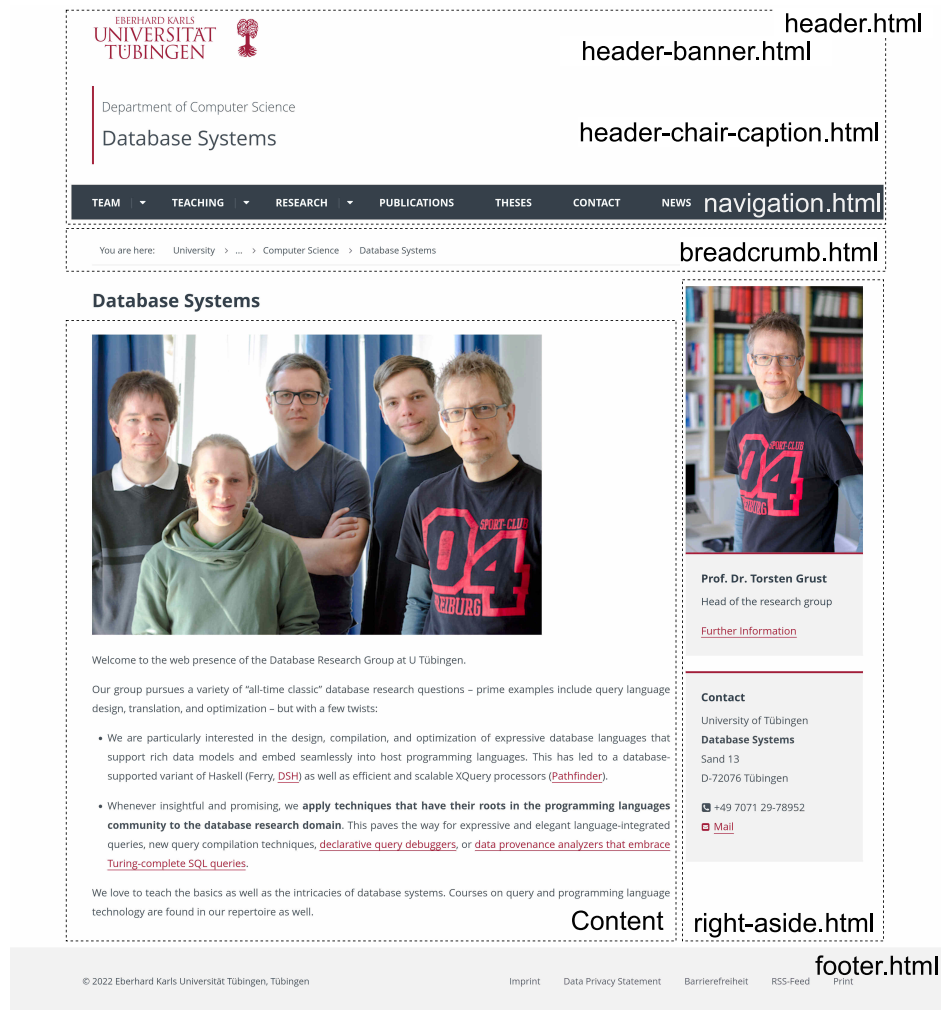


Figure 5.2: The home page rendered on a desktop device. One can see the different partials that are part of the base layout. The content of the Markdown file is rendered in the box marked with "Content". In the base layout there is also an aside box to the left of the content, which is not rendered on the home page.

Figure 5.2 shows the structure of the base template as it is rendered on large screens. The header contains the logo of the University which is a link to the home page of the University. Underneath, there are links to the department of computer science and to the chair of database systems. Next, there is the menu in blue and lastly the breadcrumbs. The main part of the page contains the content and optionally a sidebar on the left and an HTML aside box to the right. The sidebar to the left is not visible in Figure 5.2 because it is not used for the home page. But on almost every other page there is the `left-sidebar.html` (See Section 5.1.2).

The box that is marked as "Content" contains the compiled Markdown from the content file.

At the bottom of each page, there is the footer which contains links to the imprint, the Really Simple Syndication (RSS) feed, the privacy statement and some other things.

Figure 5.1 depicts the home page rendered on mobile. There, one can see that the mobile layout has some differences to the desktop layout: Instead of a first level navigation bar, the menu is hidden. It is opened with the menu button in the top right corner. Furthermore, the `left-sidebar.html` is hidden. On small screens, the `right-aside.html` box is moved to the very bottom of the page in full width.

5.1.1 Menu

Front Matter:

- `redirectToFirstChild`:

The menu item of this page will link to its first child. This is used when a page should not be accessed (because it has no content). For example used on `/research/current-projects`.

- `noMenuChildEntries`:

If set on a page, all of the children of this page will not appear in the menu. Used for `/teaching/archive`

Partials:

- `navigation.html`

Renders the menu.

Uses Front Matter: `redirectToFirstChild`, `noMenuChildEntries`

Navigating to the different parts of the website is made possible through the main menu. Since the directory structure of the content represents the structure of the website, the menu is built from it.

The menu has two front matter parameters, which can be set for any page. The first one `redirectToFirstChild` is for pages that do not have an actual web page. For example, `/research/current-projects` does not have any content. When clicking on a menu entry whose page has `redirectToFirstChild` set to `true`, one is redirected to the first child of that page.

The other parameter is `noMenuChildEntries`. It is for sub pages that have too many children to be displayed in the menu. The lecture archive at `/teaching/archive` is a good example. Lectures under the archive do not get displayed in the menu because there are around 100 of them.

In Figure 5.3, there is the opened menu for desktop and mobile. The desktop version has 3 visible layers. Each of the layers is visible in the menu except for pages that have a parent

page that is marked with `noMenuChildEntries: true`.

The mobile version of the menu only shows one layer at once in full screen. This is due to the limited space on mobile devices. Users can navigate down the hierarchy with the greater-than buttons, and back with the back button.

The partials that produce the menu are `navigation.html` and for the recursive calls `navigation-entry.html`.

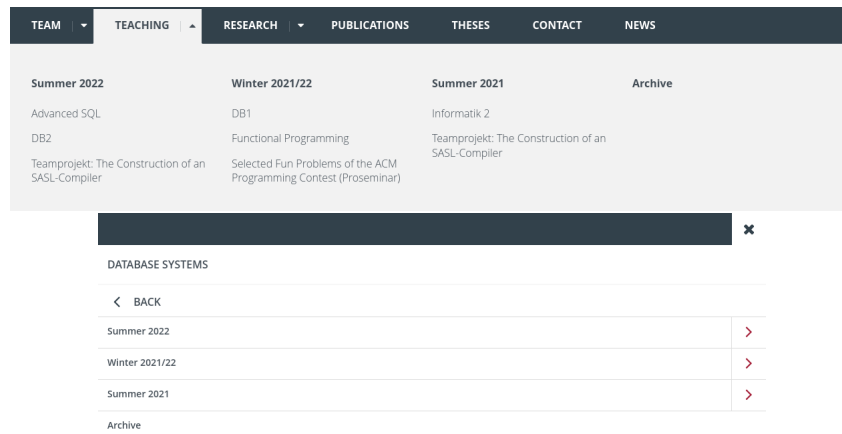


Figure 5.3: The menu on desktop (top) and mobile (bottom). The mobile version is showing the menu entries under `/teaching`

5.1.2 Section Menu

Front Matter: Same as for menu. See Section 5.1.1

Partials:

- `left-sidebar.html`

Renders the section menu to the left.

Uses Front Matter: `redirectToFirstChild`, `noMenuChildEntries`, `year` from a publication or thesis, `assignedTo` from a thesis

Most pages have a section menu to the left of the content. It is visible for breakpoint `md` (768px) and up. The section menu gives the user an overview of the section as well as a quick navigation. Generally, the section menu is rendered from the folder structure of the section. An example of this concerning the research section can be seen in Figure 5.4. The research section contains each research project of the chair, grouped in current projects and past projects.

This works in most cases, but in sections like publications or theses, the section menu is slightly different: These sections have over 100 child pages, which would blow up the section menu extensively. Instead, the sidebar contains a list of every year in which there was

a publication (a thesis). These entries are anchor links that link to the group of publications (theses) that were published that year. The groups are on the list pages ([publications](#), [theses](#)). For this to work, there is a front matter parameter `year` which can be set on publications and theses (See more in Section 5.3.3, Section 5.3.5).

Pages that contain a section menu are: [/team](#), [/teaching](#), [/research](#), [/publications](#) and [/thesis](#).

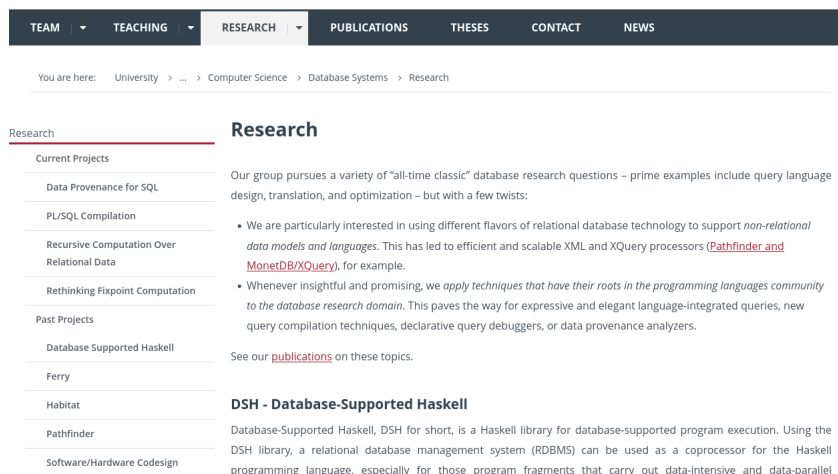


Figure 5.4: On the left, there is the section menu for [/research](#). It contains all child pages.

5.2 Content Structure

There are several data models contained in the site. Each of the sections contains its own data model except for the contact section. In Hugo, there are different ways to implement data models. The most common way is to have one data set per page. The data is declared in the front matter of the page.

Relations between the models are implemented as Hugo indices. More information concerning the implementation of the relations is found in Section 5.5.

Figure 5.5 depicts a diagram showing the different data models and their relations to each other. The most important relation is the `person` relation. Almost all models use it. A lecture (found under [/teaching](#)) is held by one or many people. A publication has one or many authors and belongs to one research project. A thesis has one or many authors. A news article can belong to one or many or no lectures.

It is important to note that the consistency of the relations is not checked by Hugo, since it does not know about them. For Hugo, these are more like buckets. Pages that have the same index declaration and value belong to the same bucket (e.g. `person: denis-hirn` is set for Denis' team member page and for all of the publications that he worked on). This

means that Hugo is unaware of the fact that logically, a publication belongs to a person, and not the other way around.

To warn about inconsistencies, some checks were implemented in components that depend on these relations.

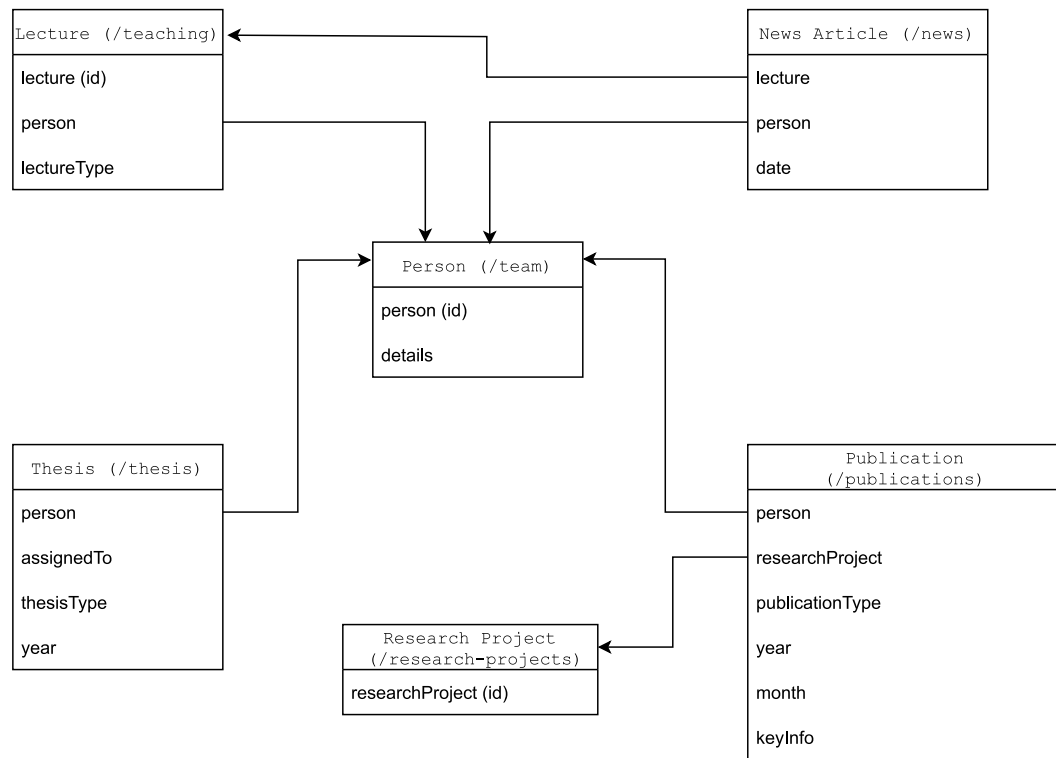


Figure 5.5: The data models and their relations. All attributes are implemented as front matter. Attributes marked with "(id)" should have one unique value as they serve as a unique id.

5.3 Section Components

Every section has its own components that display the section specific data. There are components that render one piece of data, like **single**, **details** or **summary**: The **single.html** partial of the news section renders the complete news article whereas the **summary.html** partial renders a smaller summary. Furthermore, there are list components as well that render a list of data like **list.html**.

In the following, the most important components are presented.

5.3.1 Team

Front Matter

- **indices:** **person** (id)
- **details**
An object containing information about the person.
Format: `details:{function, phone, email, address:{insitute, chair, street, city}, room}`

Page Resouces

- **portrait**
A portrait image of the person.

Shortcodes

- **team/portraits-grid.html**
Grid view of all team members with links to their pages. See Figure 5.6.
Front matter: **details.function**
Page resources: **portrait**
- **person/single.html**
The **portrait** of a person with details. See Figure 5.7.
Front matter: **details** (optional)
Page resources: **portrait**
- **person/single-picture-only.html**
Renders the **portrait** of a person.
Page resources: **portrait**

Partials

- **team/people-comma-list.html**
Renders a list of names of related persons.
Front matter: **person**
- **team/person-link.html**
Renders link to related person.
Front matter: **person.**

This section provides a great component, namely the `portraits-grid.html` partial. In Figure 5.6, the rendered component is shown with the members of the chair, their names and their function. The grid is responsive: its column count is adapted depending on the screen width. The grid tiles are links to the page of a person. The profile pictures are automatically cropped to the face of the person. This is done with a function provided by Hugo.

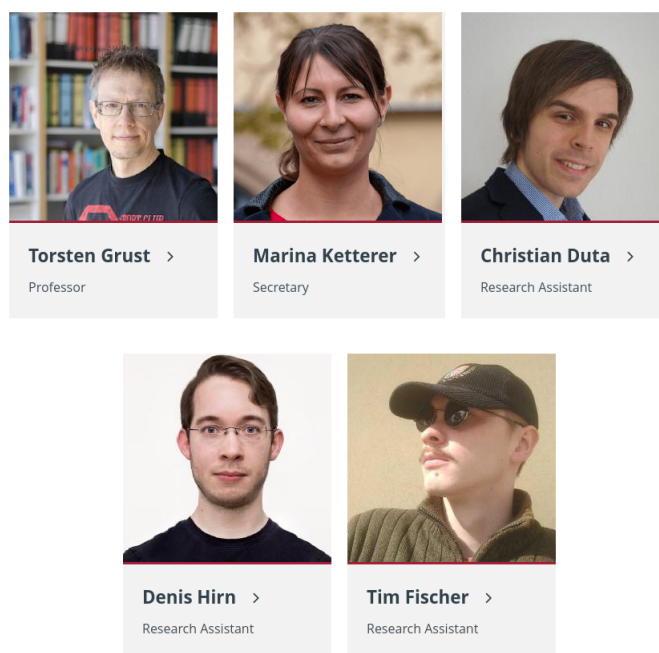


Figure 5.6: The `portraits-grid.html` showing all members of the chair with links to their pages. It appears on the team list page `/team`. It uses the page resource `portrait` and the front matter `details`

Another important component is the `person/single.html` shortcode. One can see it in Figure 5.7. It renders a profile of the person. The picture to the left has to be declared as a page resource with the name `portrait`. The information on the right comes from the `details` front matter variable. It contains the person's function at the chair and some contact information.



Figure 5.7: The `person/single.html` shortcode. It renders the **portrait** of a person and the information that is provided under the **details** front matter key.

5.3.2 Teaching

Front Matter:

- `indeces: lecture (id), person`
- `lectureType`
A string, describing the form of the lecture. (example values: `Seminar`, `Lecture`, ...)

Page Resources:

- `lecture-image`
A characteristic image of the lecture.

Shortcodes:

- `teaching/single.html`
A lecture overview, see Figure 5.10. Uses `team/people-comma-list.html` partial. Inner content is passed as tag content (Markdown).
Front matter: `lectureType`
Page resources: `lecture-image`

Partials:

- `teaching/semester-summary-grid.html`
Renders a grid for a semester page with links to the lectures. Renders `title` if `shortTitle` is not present.
Front matter: `shortTitle` (optional)
Page resources: `lecture-image`
- `teaching/list.html`
Renders the `teaching/semester-summary-grid.html` for all non-archived semesters.
- `teaching/semester-list.html`
Renders `semester-summary-grid.html` for a semester page.
- `teaching/archive/semster-summary-grid.html`
A grid of the lecture titles of one semester in the archive. Takes lecture pages as parameter.
- `teaching/archive/list.html`
Renders summary grid for each semester in the archive. Used on the `/teaching/archive` page. See Figure 5.9.

The list page of the teaching section renders an overview of the lectures from the recent semesters (all non-archived semesters). Similar to the `team/portraits-grid.html` component, each page of the recent semesters has a lecture image that is rendered above the name of the lecture. Figure 5.8 shows the rendered `semester-summary-grid.html` component. It renders a quick view of the lectures of one semester. Some lectures have really long names, which would blow up the grid-view. To avoid these long names, one can specify a `shortTitle` front matter parameter for the lecture. If it is set, it is used in the grid tile instead of the `title`.



Figure 5.8: The rendered `teaching/semester-summary-grid.html` shortcode. It gives an overview of a semester. At the top of a tile is the `lecture-image`. At the bottom is its `title` or `shortTitle` (if it is set)

This component is also used on the list pages of the not-archived semesters.

On the list page of the teaching section (`/teaching`), there is the `semester-summary-grid.html` rendered for the latest semesters which are directly under `/teaching`. Archived lectures are to be found in the lecture archive under `/teaching/archive`. When a new semester approaches, one can create a new folder for it under `/teaching`. The `semester-summary-grid.html` component automatically renders the newly created lectures under the new semester as well. If an old semester is archived, which means that it is moved to `/teaching/archive`, it does not appear on the `/teaching` list page anymore but in the archive.

The archive has a list view that renders all archived lectures by semester. In Figure 5.9 one can see an extract of the archive list page.

Lectures have several different front matter parameters: There is the `lectureType`, which is an arbitrary string. It describes the type of the course. This could be something like `Lecture` or `Seminar`.

On the lecture single pages, the `teaching/single.html` shortcode is rendered. Figure 5.10 illustrates it the "DB 2" lecture from 2022.

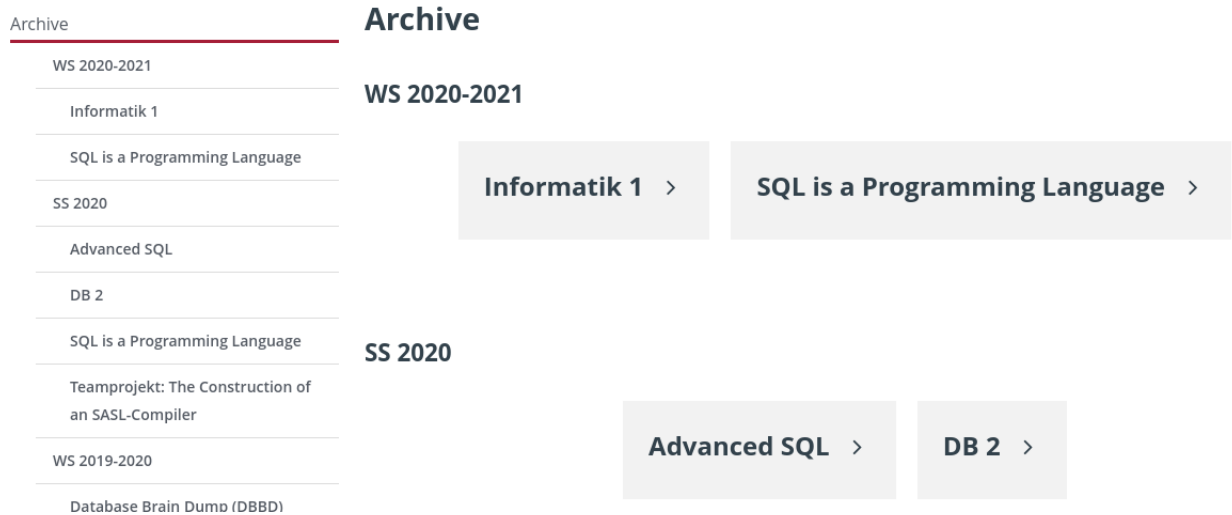


Figure 5.9: A cropped view of the `/teaching/archive` page. On the left there is the archive side bar. It shows all archived semesters with their lectures. On the right the `title` of each lecture is rendered as a link.

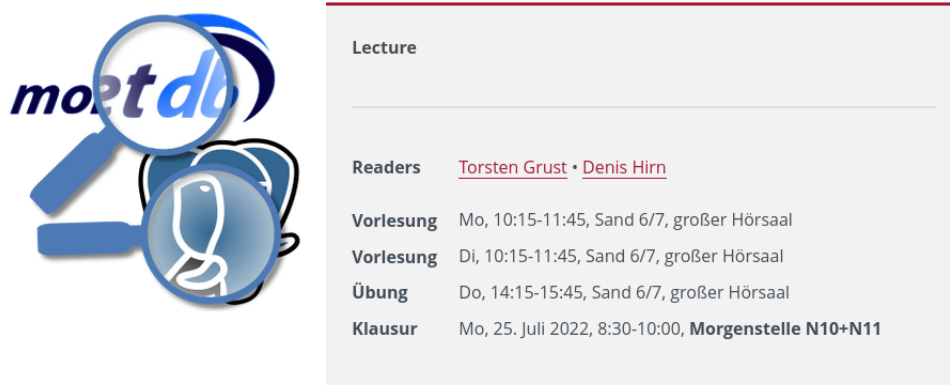


Figure 5.10: The rendered `teaching/single.html` shortcode. The header is the `lectureType` front matter. The body is passed as the body in Markdown. The picture on the left is the `lecture-image` page resource. This shortcode is rendered on each lecture page.

5.3.3 Publications

Front Matter:

- `index`: `person`, `researchProject`
- `publicationType`: Used for grouping. Values: `paper` or `report`.
- `year`, `month`: Year and month of publication. Used for sorting.
- `keyInfo`: Key information about the publication in Markdown.
- `additionalLinks`: Array of link objects with `src` and `title`.

Page Resources:

- `paper-pdf`: The PDF version of a publication.
- `poster`: The poster that belongs to a publication.

Shortcodes:

- `publications/info-box.html`
Uses `publications/details.html` partial to render a summary box like in Figure 5.11, but without publication title and buttons. Used on single pages.

Partials:

- `publications/details.html`
Renders all information about a publication. This partial is never used directly, but integrated in the `info-box.html` and `summary.html` components.
Front matter: `paperLink`, `additionalLinks`
Page resources: `paper-pdf`, `poster`
- `publications/summary.html`
Summary of a publication with its title.
- `publications/list.html`
Renders list of all publications, see `/publications` page in Figure 5.11.
- `publications/by-project.html`
List of publications that belong to one `researchProject`. Used on `/research` single pages.
- `publications/recent-list.html`
List of publications by a `person`. Used on team member pages.

In the publications section appear all publications of the chair. At the section root `/publications` there is the list-view of the publications as seen in Figure 5.11. The publications are presented in two different categories: publications and reports (front matter parameter `publicationType` with values `publication` or `report`). Reports appear at the bottom of the page because they are older and there have not been many new ones. The large majority of publications are papers. These are listed starting at the top of the page. To get a better overview of the papers, they are grouped and sorted by the publication date (front matter parameters `year` and `month`). The author is defined with the `person` key. It references a team member.

Publications

Papers

2022

Another Way to Implement Complex Computations: Functional-Style SQL UDF

[Christian Duta](#)

Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA 2022), collocated with SIGMOD 2022, Philadelphia, PA, USA, June 2022. To be published.

[Learn More](#)

[PDF](#)

[YouTube](#)

Data Provenance for Recursive SQL Queries

[Tobias Müller](#) • [Torsten Grust](#) • [Benjamin Dietrich](#)

Proceedings of 14th International Workshop on Theory and Practice of Provenance (TaPP 2022), collocated with SIGMOD 2022, Philadelphia, PA, USA, June 2022.

[Learn More](#)

[PDF](#)

Figure 5.11: A cropped view of the `/publications` page. There are two publications, rendered using the `publications/summary.html` partial

One can define some information about the publication, like the submitted conference et cetera with the `keyInfo` parameter. The `keyInfo` is defined as Markdown.

The publication can be assigned to a research project with the `researchProject` parameter. It should reference an existing project in the research section.

A PDF of the paper can be defined with a page resource that has the name `paper-pdf`. A link to the PDF is rendered in summary.

With the `additionalLinks` parameter, one can reference any external resources. These links are rendered next to the 'Learn More' button.

The 'Learn More' button is a link to the single page of the publication. The pages for each publication contains a `details.html` box that is the same as the `summary.html` but without the publication title. This is because the title is already rendered as the page title above the box.

Two other sections use publications as well: team and research. On the team member pages the recent publications of the person are rendered using the `publications/recent-list.html` partial. Each research project renders its publications with the `publications/by-project.html` partial.

5.3.4 Research

Front Matter indices: <code>researchProject</code> (id)

The research section contains all current and past research projects of the chair. Other than the description of the project, each project has a list of its publications. The partial that is rendering the list is the `publications/by-project.html` partial. Each research project has a `researchProject` front matter parameter. Publications that are marked with the same `researchProject` value appear in the list of the project.

5.3.5 Theses

Front Matter:

- **person** (index): The supervisor. Can be team member or any string.
- **thesisType**: String describing the type of thesis (e.g. Bachelor Thesis).
- **assignedTo**: Name of student.
- **year**: Year of submission.

Page Resources:

- **pdf**: PDF of thesis.

Shortcodes:

- **theses/info-box.html**
Renders a box with all essential information about the thesis. Used on `/theses` single pages.
Front matter: **thesisType**, **assignedTo**, **person**
Page resources: **pdf**

Partials:

- **theses/list.html**
Renders list of all theses categorized into
Open Thesis: **year**, **assignedTo** not set
Thesis in progress: Only **assignedTo** set to name of student
Finished thesis grouped by **year**: Both **year** and **assignedTo** set
- **theses/single.html**
Renders a summary of a thesis. Uses same *page resources/front matter*, as **theses/info-box.html** partial above.

The theses section gives visitors information about past, open, and assigned thesis topics. The structure of this section is very similar to the publications section. The list page renders a short summary of each thesis topic. Each summary has a link to its own page and a button to download the thesis (if the **pdf** was set).

The theses are grouped by their state:

- Open thesis topics appear at the top of the page. These do not have **assignedTo** set and also no **year** parameter set.
- Pending thesis topics do not have a **year** set but they are assigned to someone. Therefore the **assignedTo** parameter has the name of the student set. This parameter can

have an arbitrary string and has nothing to do with the **person** parameter.

- Finished thesis topics have the **assignedTo** parameter set and a **year** in which they were finished. All past theses are grouped by year underneath the open and pending thesis topics.

A PDF can be attached to a thesis. For that, the page resource **pdf** has to be specified.

With the **person** parameter one can give the thesis one or more supervisors.

A **thesisType** should be added to a thesis, which would probably be either **Master Thesis** or **Bachelor Thesis**. But generally it can be any string value. The section specific components are very similar to the ones from the publications section.

5.3.6 News

Front Matter:

- **indices:** **lecture** (id), **person:** the author.
- **date:** Date of publishing (yyyy-mm-dd)

Page Resources:

- **icon:** An icon that is used in the RSS feed as enclosure. RSS readers can display it.

Shortcodes:

- **news/list-single.html**
Renders each news item of a lecture. Sorted by date.
Front Matter: **lecture, date**
- **news/list-summary.html**
Renders all news items. Sorted by date. Used on home page and **/news**.
Front matter: **person, lecture, date**

Partials:

- **news/news-box.html**
Renders a summary of the news article.
Arguments: **link, author**
- **news/single.html**
Renders the whole news article. Uses **team/person-link.html**.
Front matter: **person, date**
- **news/summary.html** Renders a summary of a news article. Uses **news-box.html**
Front matter: **person, lecture** (optional)

Occasionally, the chair has to publish information about various things like, for instance, events which is done in the news section. It renders a summary list of all news articles, beginning with the latest. A news article has an author that is declared with the `person` key. In this way a link to a team member can be set. A `date` formatted as `yyyy-mm-dd` determines the publishing date by which news articles are sorted.

News articles oftentimes give important information about a specific lecture. To be able to see all news articles related to one lecture, one can specify a `lecture` front matter value that refers to a lecture which has the same value. The `news/list-single.html` shortcode renders the related news articles for a lecture. It is used on each lecture single page.

On the list page of the news section (`/news`) the `news/list-summary.html` shortcode is rendered. It uses the `news/summary.html` partial to render a summary of each news article. In Figure 5.12 the rendered component is shown.

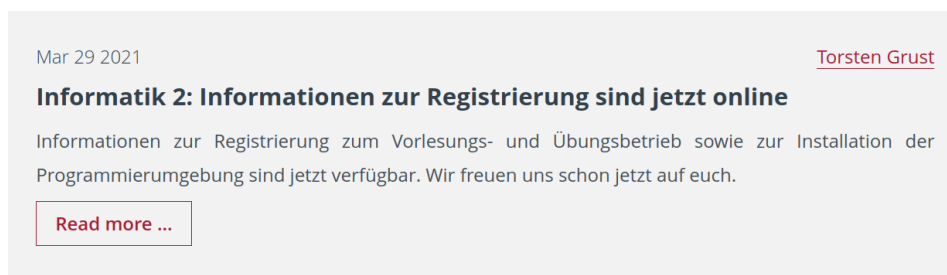


Figure 5.12: `news/summary.html` partial. It shows the author, the date of publication and a summary of the article.

The website has a template for rendering an RSS feed as well. It is found under `layouts/_default/rss.xml`. The template iterates over all news articles and renders their content as the description. It is also possible to specify an icon for the news article. If present, the icon is used in the RSS feed as an enclosure. RSS readers can use the icon if they want to. The icon has to be declared as a page resource with the name `icon`.

5.4 Additional Components

- `partial.html`

This shortcode is an adapter to render partials in content files. In Hugo, one can only use partials in template files and shortcodes in content files. If one wants to use the same component in both template and content files, one would have to create a shortcode and a partial for it. This code duplication should be avoided. With the `partial.html` shortcode one can instead define the component as a partial to use it in template files. To use the component in content files one can call it through the `partial.html` component. There is an example in Figure 4.8, line 21.

- `html.html`

This shortcode is for rendering pure HTML in markdown files. The HTML is passed as the body.

- `title-link.html`

It Renders a link into a heading and is used on `/research` page to link to research projects.

- `carousel.html`

This is a basic HTML/CSS carousel. The carousel accepts a page resource name as argument. In Figure 5.13 there is a usage example of the carousel. Figure 5.14 shows the rendered component.

```
1 resources:
2   - src: 'carousel*'
3     name: image-:counter
4 ---
5 {{< carousel "image-**" >}}
```

Figure 5.13: `carousel.html` shortcode usage example. All files starting with “carousel” will be added as a page resource with the name “image-:counter”. Hugo replaces the `:counter` variable with ascending numbers. The `carousel.html` component is given the name of the page resource, that defines the images. The double star “**” is a wildcard, so this collects all images defined in the page resource.

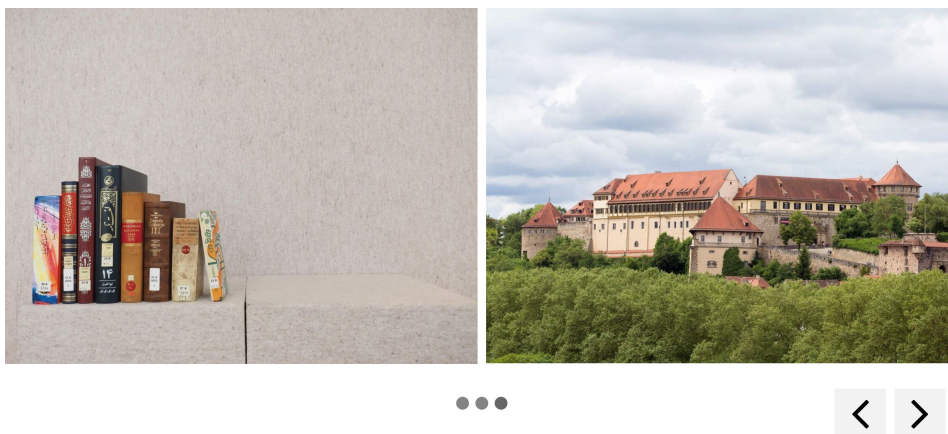


Figure 5.14: The rendered `carousel.html` shortcode on desktop. In the bottom right corner, there are the navigation buttons. At the bottom, the dots represent the number of total images. Only one picture is displayed at a time on small screen sizes.

5.5 Build Environments

Some of the functions of the website can be configured with configuration files. Those reside in the `config` directory. The `config/_default/config.yaml` file contains all configurations that apply to every build environment. The production environment is used when running the `hugo` command. The development environment is used when running the development server with the `hugo server` command.

In the default configuration the `baseUrl` is set which is prepended to links, created with the `ref` or `Permalink` function. In production this is the domain where the website is hosted. In the development environment, the website is running locally so this value has to be different. This is the purpose of the `config/development/config.yaml` file which has as its only entry a `baseUrl` of `/`, so that the website can be opened at `localhost:1313`.

In the default configuration all services like twitter or instagram are deactivated. If one wanted to embed tweets or instagram posts in the future, these would need to be activated first.

One of the most important entries are the related indices. There are three of them: `researchProject`, `person` and `lecture`. These indices are used to access the related content of pages as depicted in Figure 5.5. To create a relation between pages, one has to declare the same value for an index, each member, for example has the `person` index set with their name as value. To assign a publication to that person, one has to declare the `person` index for a publication with the corresponding name: `person: <some-name>`.

Hugo provides the `Site.RegularPages.RelatedIndices <page> <index>` template function. This functionality is used for example in the `publications/summary.html` component in Figure 5.11. This is where the authors of the paper are calculated using the `RelatedIndices` function.

5.6 JS/CSS compilation

As explained in Chapter 2, one prerequisite for the website was to implement it using the styles of the cooperate design of the University. These styles are contained in the `assets/css/page.css` file. This file was just copied out of the browser so that if the University changes their styles, one can download them and exchange the old stylesheet with the new one.

The `assets/sass/custom.sass` file contains custom styles that were added to the cooperate design. It contains - for instance - style rules for the team grid in Figure 5.6.

The JavaScript (JS) files from the University cannot be used, because they contain a lot of unnecessary code like the cookie banner. Because the file is minified, one cannot extract parts of the code to use it for this site. Therefore the logic for the menu was reimplemented, which is the opening and closing of the menu on desktop and mobile. This logic is found

in the `assets/js/page.js` file. The `top-scroller.js` file contains the logic for the “scroll-to-top” button. The button appears upon scrolling a short distance down a page. Finally the `carousel.js` file contains all logic that is needed for the carousel component. The script in Figure 5.15 handles all the JS asset bundling. All JS files are concatenated to one file. The resulting file is minified to reduce download size. Then it is fingerprinted for cache busting. The JS file is embedded in the site in the `scripts.html` component. The process of JS bundling is similar to the CSS/SASS asset bundling.

```
1 {{ $page := resources.Get "js/page.js" }}
2 {{ $topScroller := resources.Get "js/top-scroller.js" }}
3 {{ $carousel := resources.Get "js/carousel.js" }}

4 {{ $js := slice $page $topScroller $carousel | resources.Concat "js/
   main.js" | minify | fingerprint }}

5 {{ return $js }}
```

Figure 5.15: `func/compileJs.html` partial. JS is bundled, minified and fingerprinted

5.7 Archetypes

Archetypes in Hugo serve as templates to create new page content. They can be used from the command line as explained in detail in Section 4.7. There are archetypes of each type of data that is present in the site: member, alumni, lecture, publication, thesis, news.

Conclusion

In this work the concept of SSGs was explored and tested by implementing a real website. Using the SSG Hugo, the website of the Database Systems chair of the University of Tübingen was reimplemented. The website was realized according to the cooperate design guidelines [5] of the University.

A few downsides of Hugo transpired, during the process:

First of all, debugging is sometimes hard in Hugo, because the error messages are difficult to interpret. This drawback becomes apparent when implementing a more complex algorithm. Apart from error messages, the only debugging tool that Hugo supplies is a console logger function. To be fair, most of the time one does not implement very complex algorithms in a Hugo website.

The other downside is the fact that Hugo does not provide lambda functions and higher order functions like map or filter. This is a minor issue but having these functions available would be more convenient at times.

The website itself could also be improved in some ways:

Firstly, one could let Hugo run in a Docker container [8]. In doing so the environment (Hugo, go, rsync) would be abstracted from the server, thus making its maintenance easier.

Secondly, the website contains one huge CSS file with 16,000 lines that was copied from the website of the University. It would be better to have the CSS split up in SASS files. Unfortunately, it was not possible to get a hold of the original non-compiled style sheets from the University. The only advantage of using the compiled CSS file is that it can be just replaced with the new version if the University makes style changes for some components. Finally, either Hugo or oneself could implement more sophisticated data model relations with relation integrity.

Other than in these few instances where the website could be improved, the website is a total success. Hugo build times are really fast (on a mediocre laptop: 3 seconds - 372 pages). The deployment pipeline is very fast and easy to use. Content management is quite easy since content is written in Markdown and data is declared in YAML Ain't Markup Language (YAML) as front matter. With the cooperate design, the website has a great look. Hugo is simple enough to have people learn it quickly and at the same time complex enough to implement important functionality. These traits make it a great framework. Enjoy!

Bibliography

- [1] internetlivestats.com. *internet life stats*. URL: <https://www.internetlivestats.com/total-number-of-websites/>. (accessed: 05.09.2022).
- [2] DB Systems Group. *Database Systems Website*. URL: <https://db.inf.uni-tuebingen.de/>. (accessed: 05.09.2022).
- [3] Jason Long Scott Chacon. *Git*. URL: <https://git-scm.com/>. (accessed: 05.09.2022).
- [4] John Gruber. *Daring Fireball: Markdown*. URL: <https://daringfireball.net/projects/markdown/>. (accessed: 09.09.2022).
- [5] Universität Tübingen. *Cooperate Design - University of Tübingen*. URL: <https://uni-tuebingen.de/en/einrichtungen/verwaltung/stabsstellen/hochschulkommunikation/service-fuer-beschaeftigte/corporate-design/startseite/>. (accessed: 05.09.2022).
- [6] Bjørn Erik Pedersen. *The world's fastest framework for building websites*. URL: <https://gohugo.io/>. (accessed: 05.09.2022).
- [7] Google. *The Go Programming Language*. URL: <https://go.dev/>. (accessed: 05.09.2022).
- [8] Docker Inc. *Docker*. URL: <https://www.docker.com/>. (accessed: 05.09.2022).