# Chapter 1

# Introduction

E-commerce has been around for 20 years. Since the release of the Netscape browser in 1994 companies have been selling their products online. From 1995 to 2000 the number of companies who were selling their products online increased significantly. Venture capitalists invested huge amounts of money in these new internet companies. Sadly, the excitement wasn't well founded. The world wasn't ready for a shift from traditional retailing to e-commerce. People were wary about ordering products online, online payment methods weren't very widely excepted and parcel services weren't optimized for delivering the products to the people.

The excitement about internet companies led to the dotcom bubble bursting in 2000. A lot of new internet companies went bankrupt, others had to downsize significantly. A lot of survivors of the dotcom bubble are still around today. They have grown become huge internet companies, some examples are: amazon.com, ebay.com and dell.com.

In the last 15 years online retailing has been steadily increasing. People were getting more comfortable buying their products online, online payment methods improved significantly and parcel services optimized their distribution network. These improvements gave small companies the opportunity to start selling their products online. It is expected that the online retailing market will overtake traditional retailing within ten years.

## 1.1 What's in the course

In this course you will learn to create and manage a web store. There are a lot of platforms which allow you to create a web store. In this course we chose to use Drupal. Drupal is a content management system (CMS) written in PHP. It has many features that will make it easy to create a web store.

First we'll learn how to create, manage and edit a basic Drupal site. Afterwards we add the web store functionality through the use of modules. During the course you'll also learn how to write PHP code, manage your site through

git, collaborate on your site and deploy it to a server.

## 1.2 What's a CMS

Drupal is a content management system or CMS. The name is a good description of what it actually does. It's an application which makes it easy to manage content. The content can be of different types. For example, a blog can use a CMS. The type of content on the blog will be articles. The CMS makes it easy to create, edit, remove and manage articles. Another example where a CMS can be useful is when you create a web store. The type of content we use here are products. The CMS makes it easy to manage the products, for example: add a product, change the price or add a shipping method.

A lot of big website use a CMS as their background application. Examples include: `www.engadget.com`, `www.puma.com`, `www.societegenerale.com/`.

Figure 1.1 shows different CMS systems.

Figure 1.1: Examples of CMS sytems

## 1.3 What's Drupal

As mentioned before, Drupal (logo figure 1.2) is an open source CMS written in PHP. Drupal is web based so it uses HTML, CSS and JavaScript as application front end. A Drupal application is completely customizable. We can change the look of our site by changing the HTML and CSS (Drupal calls this theming).

The application back end can be extended by adding our own code or code other people made available online (Drupal does this through its module system).

Drupal was created in 2001 by Dries Buytaert. He got his degree in computer science at the university of Antwerp and his phd at Ghent university. In 2007 he started the company Acquia which provides services to organisations using Drupal as their CMS. In 2009 Acquia helped with the relaunch of `whitehouse.gov`. Next to whitehouse.gov a lot of other sites use Drupal. At `https://www.drupal.com/showcases` you can find a list of different sites build with Drupal.


Figure 1.2: Drupal logo

## 1.4 Drupal 8

Even though Drupal 8 is still in beta we'll be using Drupal 8 to create our applications in this course. Drupal 8 is still in beta but it has been feature frozen so no new features will be added. The Drupal core developers are now just performing some bugfixes prior to the final release. We do not expect you to encounter any of these bugs but if you do, you should report them to the Drupal community.

## 1.5 Review exercises

1. Look up the definition of a CMS on wikipedia.
2. Think of five content types that could be managed by a CMS.
3. Find five sites which use Drupal as CMS.
4. Name three other content management systems.

# Chapter 2

# Tools installation

## 2.1 Acquia Dev Desktop

The Acquia Dev Desktop provides an easy way to run a Drupal site on your local computer. The tool includes an AMP (Apache, MySql, PHP) stack which allows your Drupal application to be executed. You can use it to create a new Drupal site and manage it.

### 2.1.1 Installation

- Download the version of Acquia dev desktop for your operating system at `https://www.acquia.com/downloads`.
- When your download is complete, run the installer.
- Keep clicking Next, accept the licence agreement and select the installation location for dev desktop as well as the location where your Drupal sites will be stored. In this course we use the `C:\drupal_sites` directory but you can change this to a directory of your preference. It is advised to choose a directory that is short and has no spaces.

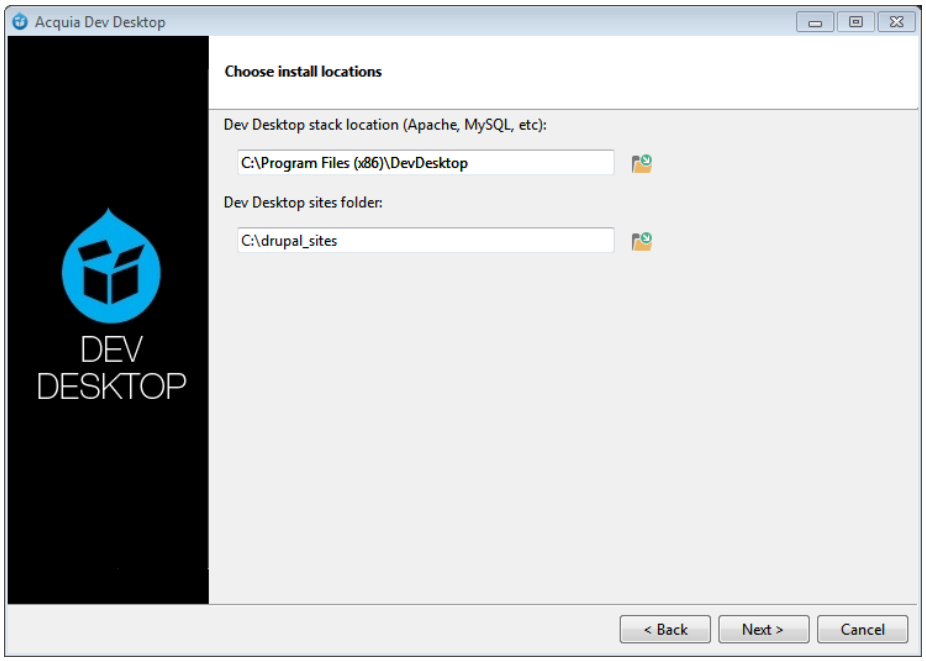



Figure 2.1: Installation location

- When the installer is complete you are ready to create your first site. Before we do that we'll install some useful tools that will help us during development.

## 2.2 NetBeans

When we are developing our Drupal site we'll write PHP, html and css code. In this course we use the Netbeans ide to do all this. You can download the NetBeans editor at: `https://netbeans.org/downloads/`. You can choose to download the version for HTML5 an PHP or the ALL version.

### 2.2.1 installation

Run the NetBeans installer. You can use the default settings or change them to your liking.

### 2.2.2 configuration

Next we'll configure NetBeans to associate the Drupal file extensions to the correct file types. For example **.module** files should be opened as a PHP file. Go to **Tools >Preferences >Miscellaneaus >Files** there you can add new

file extensions and associate the file type. Associate the following extensions with the **text/x-php5** MIME type:

- module
- install
- test
- profile
- theme
- engine

Associate the following extensions with the **text/plain** MIME type:

- info
- po

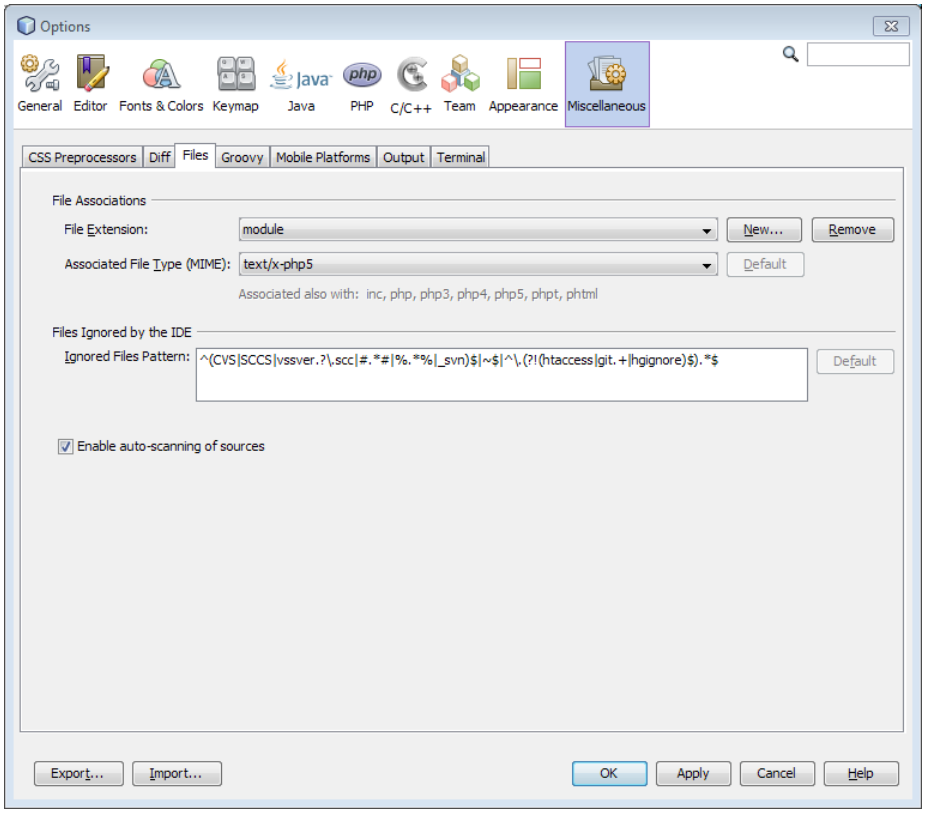



Figure 2.2: Extension configuration

## 2.3 WinLess

WinLess is a less compiler for windows. The less language makes it easier to write css code. In this course we assume you are familiar with css and less so we won't go into depth on how to write less code. To install the less compiler, download the latest version of WinLess from `http://winless.org/` and run the installer.

WinLess allows you to select a less file and automatically compile it to a css file in the directory you specify. (see Figure: 2.3)

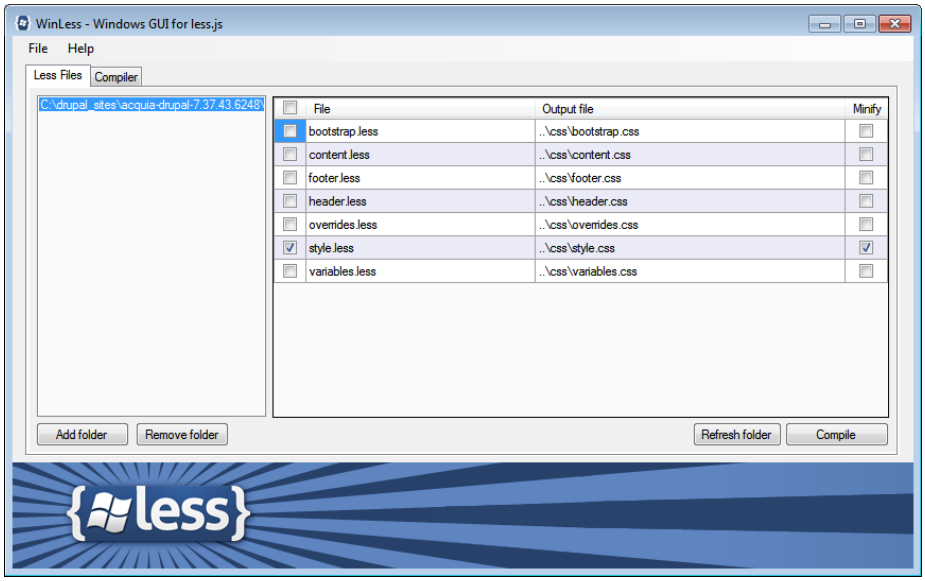



Figure 2.3: WinLess interface

## 2.4 Drush

Drush is a command line tool which makes it easy to manage your site. Since Drush is included in the Acquia dev desktop installation you don't need to install it separately. You can test your Drush installation by opening a command line window and typing **drush** and hitting enter.

## 2.5 GIT

GIT is a versioning system, it enables you to share and manage your code. You can download the git installer here: `https://git-scm.com/downloads`

### 2.5.1 Installation

- Run the installer.
- In the window **Adjusting your PATH environment** select the option **Use Git from the Windows Command Prompt**
- In the next window select **UseOpenSSH**
- Configure the line endings as **Checkout Windows-style, commit Unix-style line endings**
- Finish the installer wizard.

# Chapter 3

# Create your first Drupal site

## 3.1 Acquia Dev Desktop

As mentioned before we will be managing our local Drupal installation through the Acquia Dev Desktop tool. The tool contains a PHP hosting environment which includes: the PHP runtime, an Apache web server, a MySQL database server and phpMyAdmin for database management.

Figure 3.1 shows the Acquia Dev Desktop interface. On the left you can see an overview of the sites you created. The plus and minus signs in the bottom left corner allow you to add or remove a Drupal site. In the right panel you can see the information of the site you selected. It contains the following:

**Local site** The local url to your site. Refers to the local Apache server running on port 8083.

**Local code** Shows the path to the local codebase. All the code your site uses is stored in this location on your computer.

**Local database** A link to the phpMyAdmin page of your local database.

**PHP version** The PHP version your site uses, we will use the default version (5.5.27).

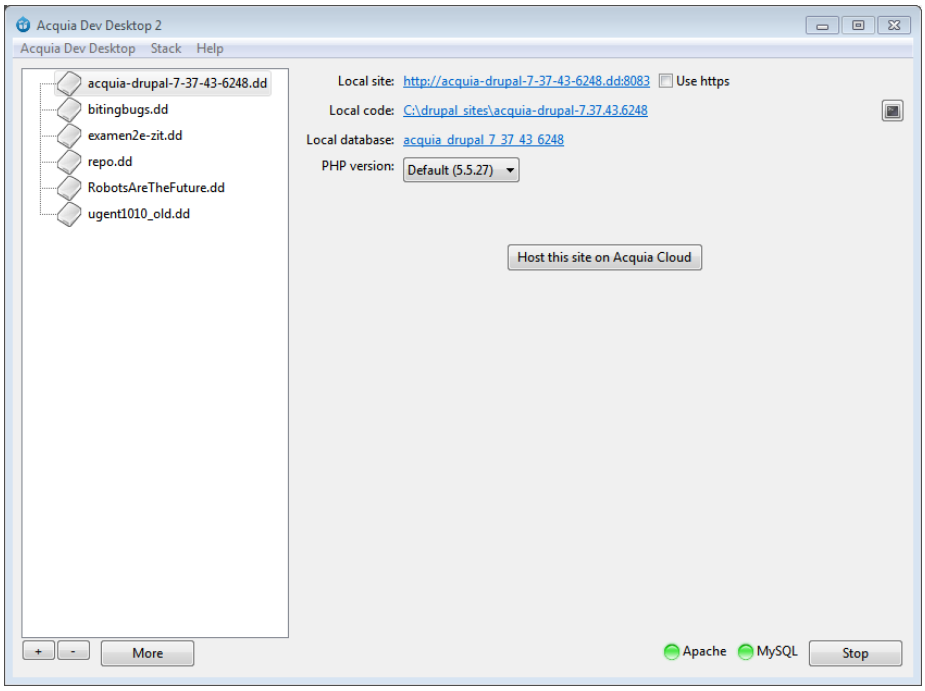



Figure 3.1: The Acquia Dev Desktop interface

## 3.2 Creating a new site

To create a new site through Acquia Dev Desktop, click the plus button in the bottom left corner. Select **New Drupal site...** in the context menu (Figure 3.2)

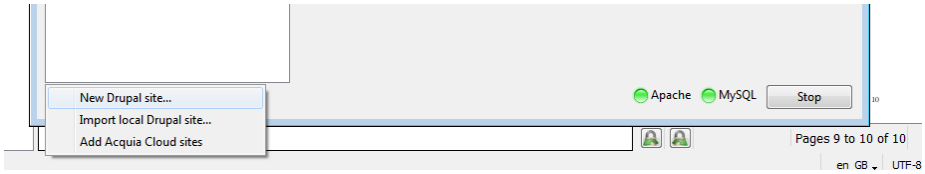



Figure 3.2: New site menu

In the next window you will see an overview of different standard Drupal installations. Select the Drupal 8 version (Figure 3.3)

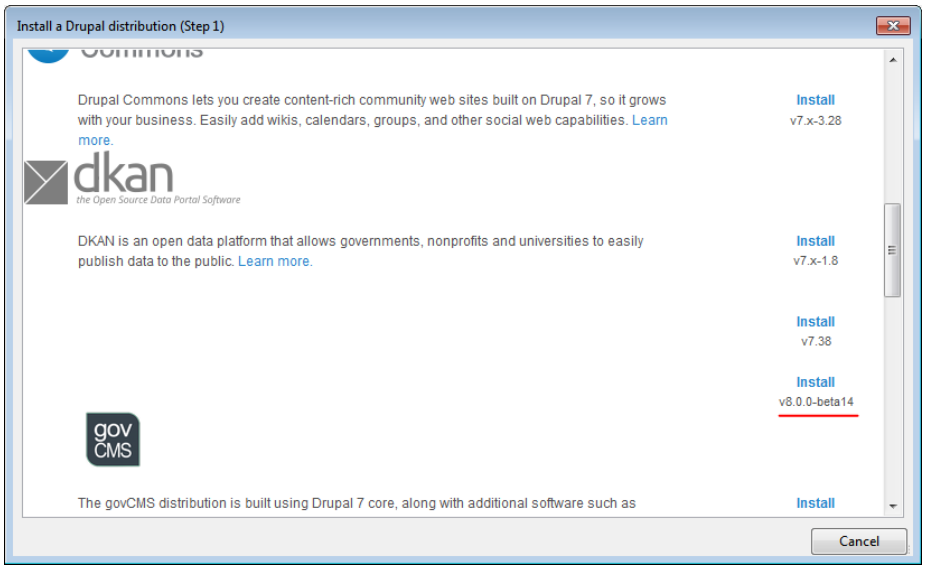



Figure 3.3: Drupal version selection

Fill out the following site information in the next window (Figure 3.4). Note that if you have configured Acquia Dev Desktop to store your sites in a different location the default path might not be `C:\drupal_sites\`.

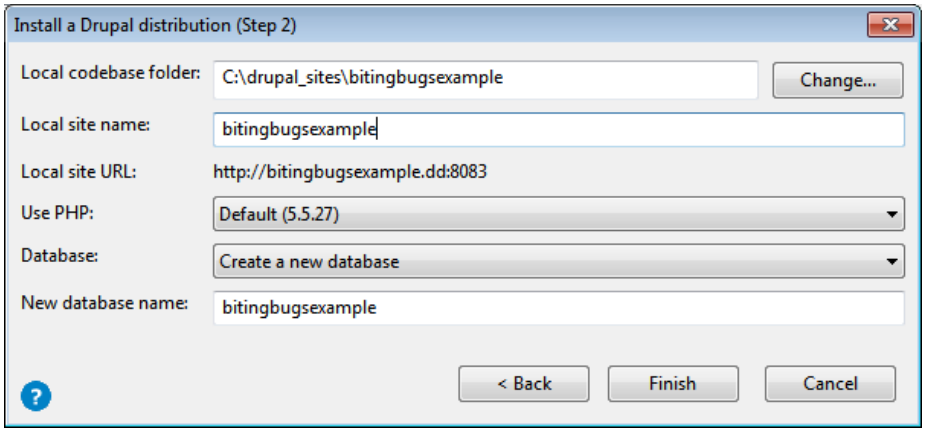



Figure 3.4: Site configuration

Click **Finish**. Acquia Dev Desktop unpacks the selected Drupal 8 archive into the local codebase folder. When the process finishes click the link to your local site in the right panel of the Acquia Dev Desktop, this will open the installation page for your site in your default browser (Figure 3.5).

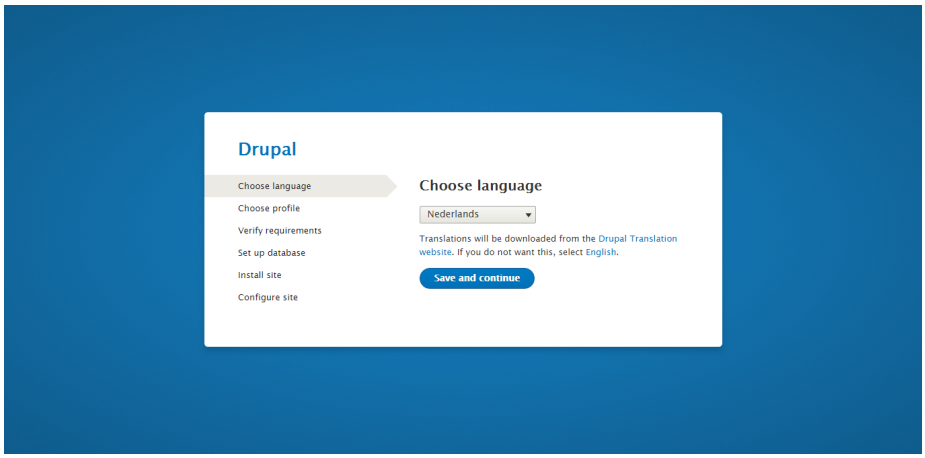



Figure 3.5: Site installation: Choose language

Since this course is in English we'll select it as our default site language. Click **Save** and on the next page select the **Standard** installation profile (Figure 3.6).

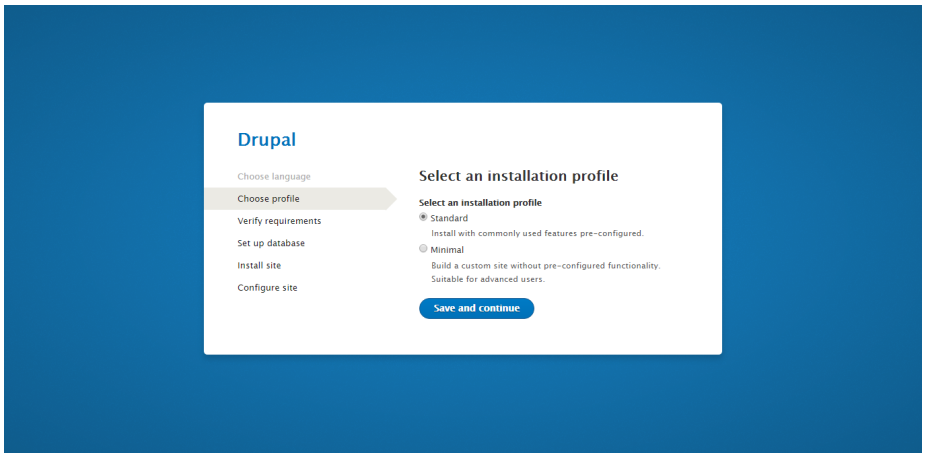



Figure 3.6: Site installation: installation profile

Click **Save and continue** This step might take a few minutes. After the installation has finished we need to configure some site properties. Enter the following information (Figure 3.7 and 3.8):

**Site name:** bitingbugsexample

**Site email address:** noreply@bitingbugsexample.com

**Username:** drupal

**Password:** Drupal

**Confirm password:** Drupal

**Email address:** info@bitingbugsexample.com

**Default country:** Belgium

**Default time zone:** Europe/Brussels

After entering the right configuration you can click the **Save and continue** button.

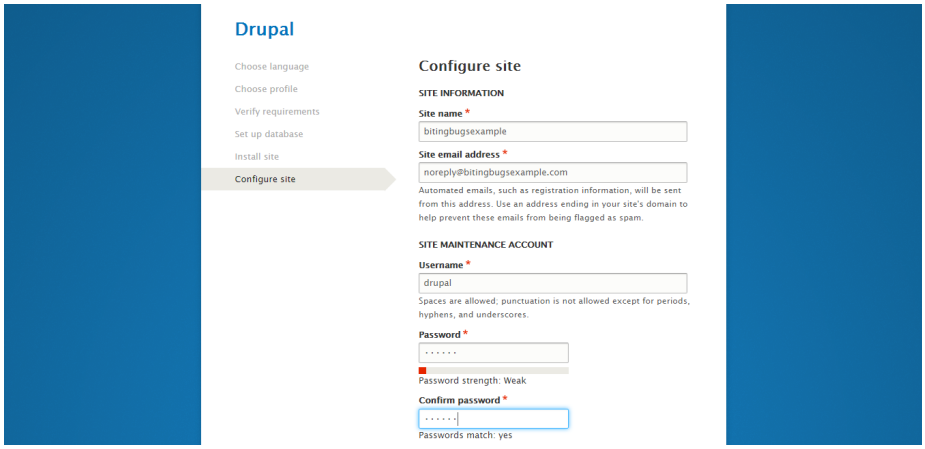



Figure 3.7: Site installation: installation configuration 1

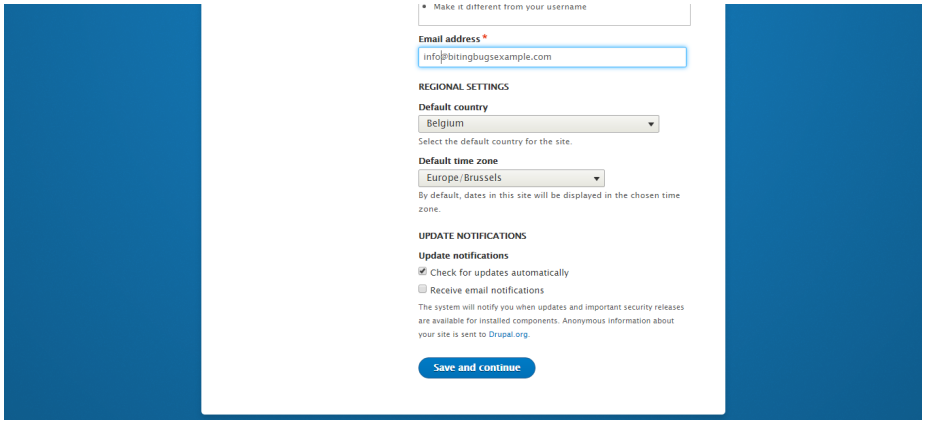



Figure 3.8: Site installation: installation configuration 2

Et voila, you have your first Drupal site. It's pretty basic, you only have a welcome page, no content yet. By default, after completing the installation, you are logged in as administrator (figure 3.9). To see the non administrator layout click the **Log out** link in the top right corner. As you can see we only have a basic one page site with our site title at the top. Now log back into your site with username: drupal and password: Drupal.

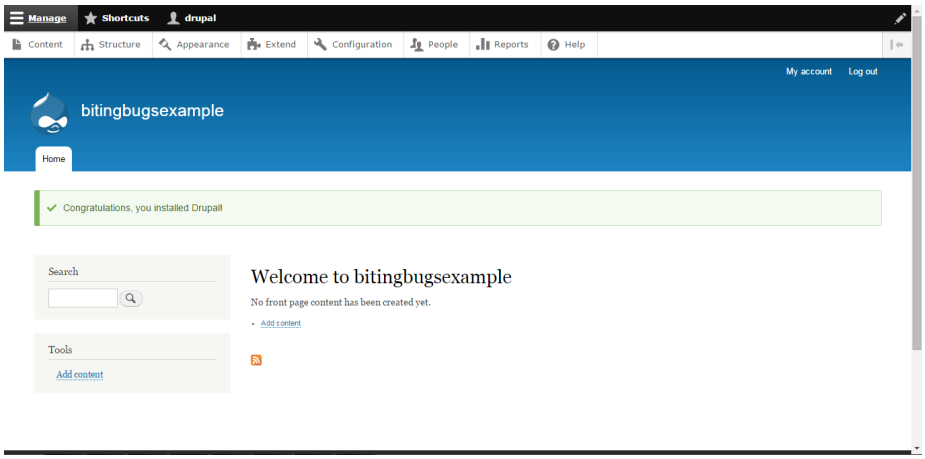



Figure 3.9: Your first Drupal site.

## 3.3 review exercises

Create a new Drupal 8 site with the following properties:

**Sitename** exploringdrupal8

**database** exploringdrupal8

**Language** English

**Username** drupal, password: Drupal

**Timezone** Europe/Brussels

**Country** Belgium

# Chapter 4

# The Drupal structure

## 4.1 Structural elements

Drupal 8 core defines eight types of structural elements. To see an overview of these elements click on the **Structure** button in the administrative menu (Figure 4.1).

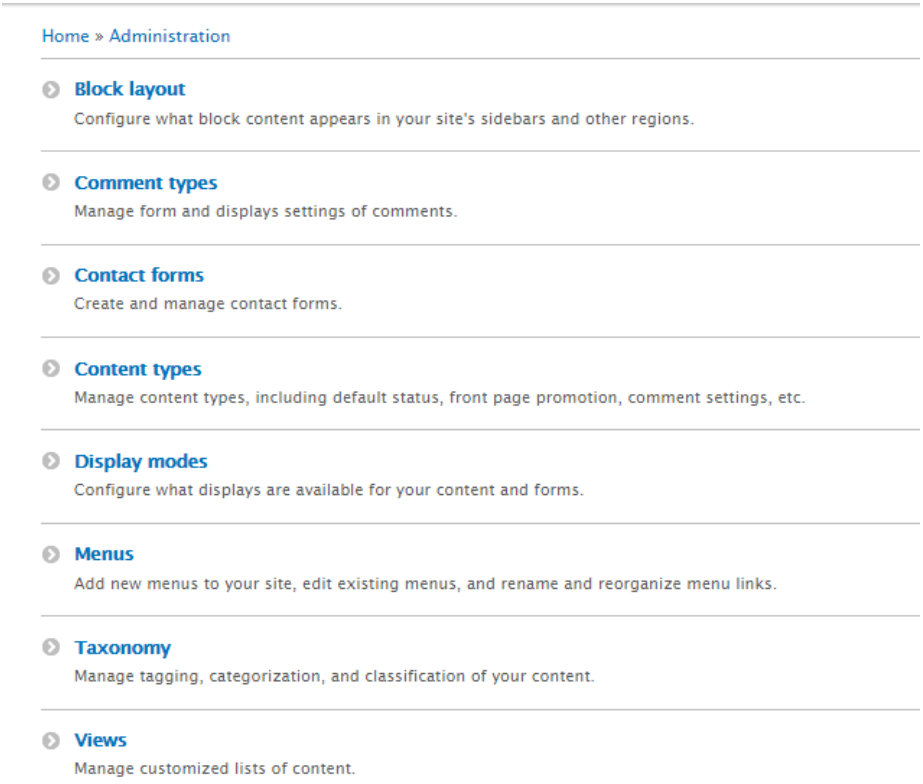



Figure 4.1: Structural Elements

Each of these elements has a certain use within Drupal. The following list describes each of the structural elements. Don't worry if not all the elements are clear to you, when you start changing and using your site the structure will become clear.

**Block layout:** Blocks are site elements which can be positioned in different places on your site. For example the search field, which is visible on the left side of your first Drupal page, is a Drupal block. You can put a block in different places on your page, these places are called **regions**. These regions depend on the Drupal theme you are using. When you click the **Block layout** link you will see the following page (Figure 4.2).

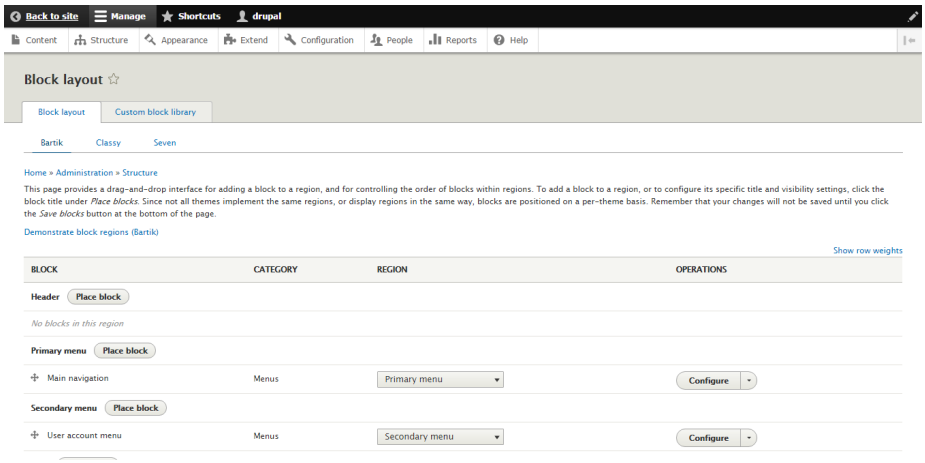



Figure 4.2: Block layout

This page shows a list of the different regions and allows you to add blocks to each of these regions. You can click the link **Demonstrate block regions(Bartik)** to view the regions of the current theme (Bartik) (Figure 4.3).

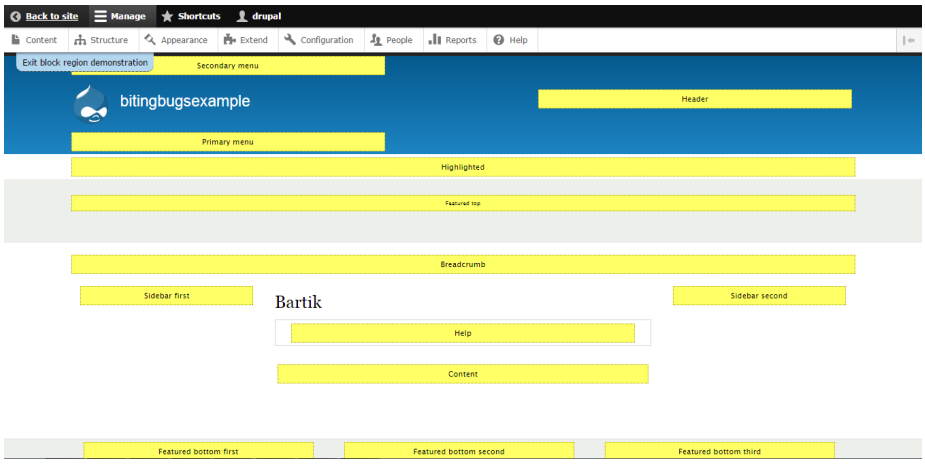



Figure 4.3: Bartik regions

**Comment types** The comment types page allows you to create and manage different comment types. When you add content to your site you can allow people to comment on the new content. The comment type defines the fields that the commenter has to fill out when he's writing his comment. The default comment type has only one field: the comment body. We

could, for example, create a new comment type which includes a field for the name and age of the commenter.

**Contact form** The Personal contact form is the form for site visitors to contact registered users; the name and recipients of this form cannot be edited. Other forms listed here are your configured site-wide contact forms, which site visitors can use to send mail to a centralized email address or addresses. You can edit the name and recipients of site-wide forms by choosing the Edit operation. You can also configure the fields and display of both personal and site-wide forms.

**Content types** The content types page is very important. The content types define which kind of information your CMS will manage. A content type has different fields. These fields define the information that is stored in the content type. By default Drupal has two content types: Article and Basic page (Figure 4.4);


Figure 4.4: Default content types

When you click the **Manage fields** button on the right you can see what kind of information is stored in this content type. (Figure 4.5). As you can see, the Article content type has four fields: Body, Comments, Image and Tags.

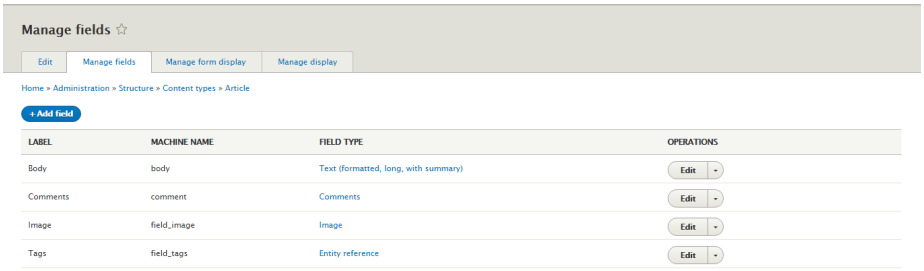



Figure 4.5: Article content type fields

Next to the **Manage fields** menu item tab you have the **Manage form display** and **Manage display** tabs. These allow you to edit which fields are displayed when an element is created/edited or viewed.

**Display modes** Display modes define different ways in which information is displayed. There are two types of display modes: form modes and view modes. Form modes are used when the content is created or edited, view modes when the content is viewed. When you go to **Display modes** → **View modes** (Figure 4.6) you will see the different ways in which a content type can be displayed.

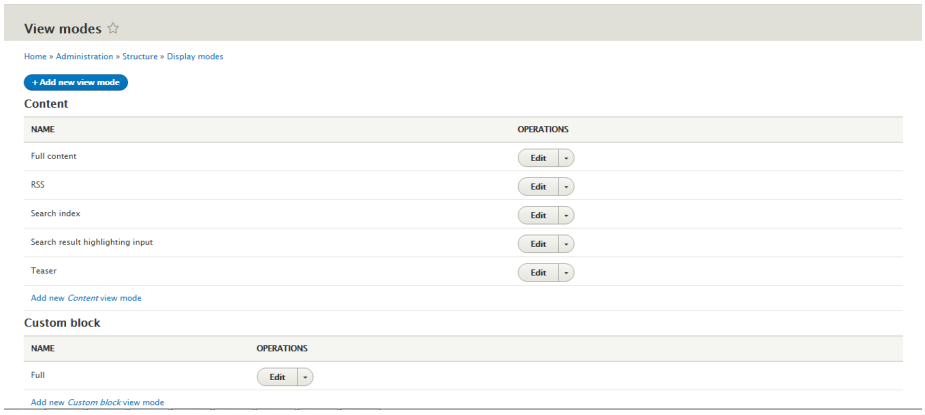



Figure 4.6: Default view modes

When you go to **Structure** → **Content types** → **Article** → **Manage display** you will see the following page:

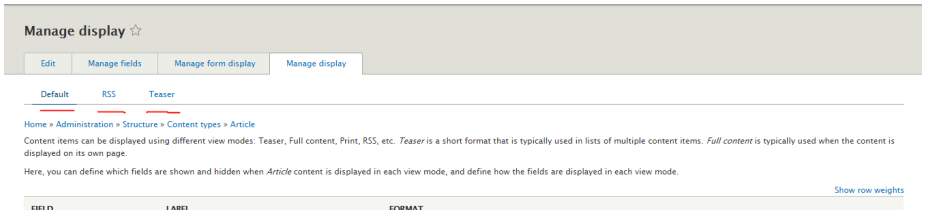



Figure 4.7: Display modes

There you see the different display modes that you can use for your Article content type. At the bottom of the page you can enable other display modes in the custom display settings dropdown (Figure 4.8).

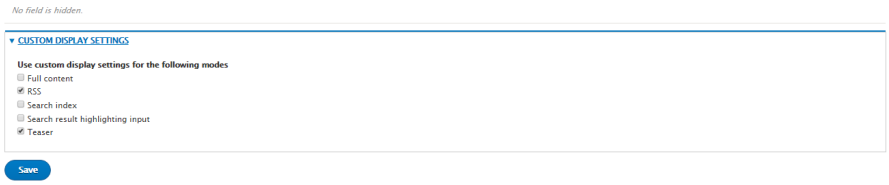



Figure 4.8: Custom display settings

**Menus** Menus are easy, they define a menu with different menu items. Each menu has a corresponding block that is managed on the Block layout page.

**Taxonomy** A taxonomy defines lists of terms, these terms can be associated with the content. A list of terms is called a vocabulary. For example: if we have a site with news articles about sports we could add a tag to each article to tell what the article is about. These tags can be defined in a vocabulary.

**Views** A view enables you to create a display based on different content types. This course has a whole chapter dedicated to Drupal Views so we won't go into details here.

## 4.2 Bitingbugs example

In the following sections we will review the content of this chapter by applying it to an example website. In the previous chapter we created the bitingbugs example website through the Acquia Dev Desktop. In this and the following chapters we will keep adding features to this site. Our goal is to create a webstore for selling edible insects. The site will also provide a database with recipes so people know how to cook with the insects.

### 4.2.1 Change the site title and logo

To make our site a little bit prettier we will change the logo and title. To change the site title go to **Configuration** → **Site information**. There change the site name to **Biting Bugs** (Figure 4.9).

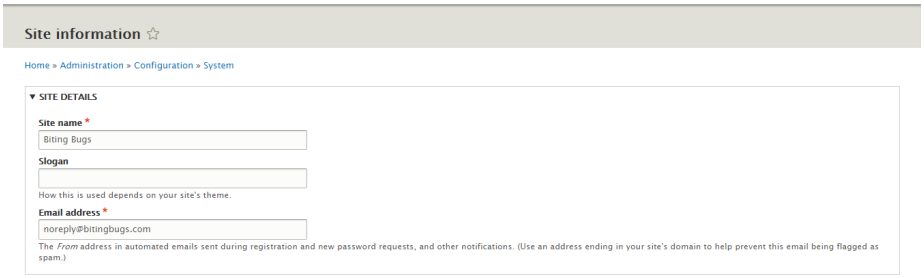



Figure 4.9: Changing the site title

The logo is part of the Drupal theme you are using. To change the logo go to: **Appearance** → **Settings** → **Logo image settings**. Uncheck **Use the default logo supplied by the theme** and upload the file `bitingbugs_logo_transp_white_right_small.png` (Available in the course files zip). Click **Save configuration**.

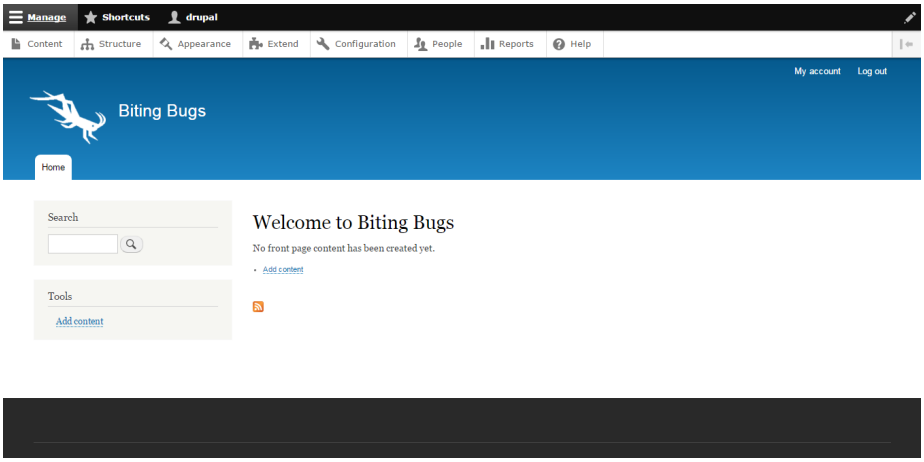



Figure 4.10: Changed logo and title

### 4.2.2 Removing the Search and Tools block

On the right side of the page we have the **Search** and **Tools** block. We don't need them for now so you can remove them by going to **Structure** → **Block layout** and moving them from the **Sitebar first** to the **None** region (Figure 4.11). Click **Save blocks**.

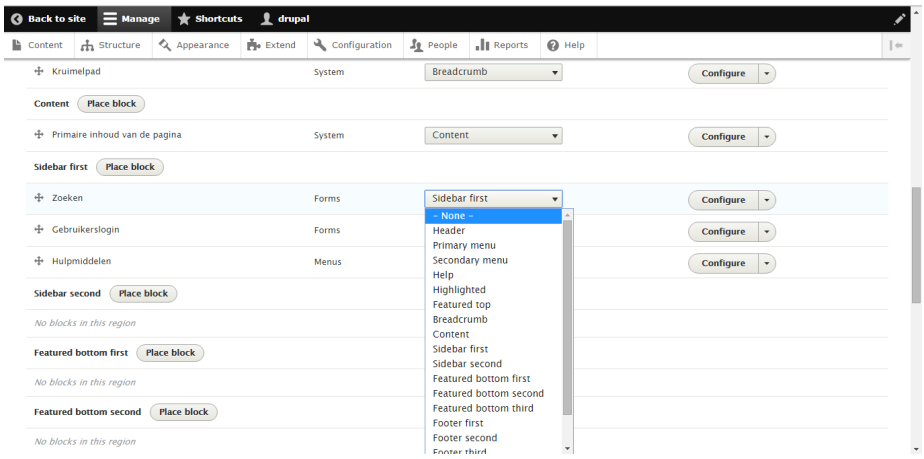



Figure 4.11: Remove a block from a region

### 4.2.3 Adding a Taxonomy

To categorise the recipes on our site we will add a taxonomy. This taxonomy will contain different types of dishes. Go to **Structure** → **Taxonomy** → **Add vocabulary**. Use the following settings:

**Name:** Dish types

**Description:** Describes the type of dish we will add.

**Vocabulary language** English

**Default language** Site's default language (English)

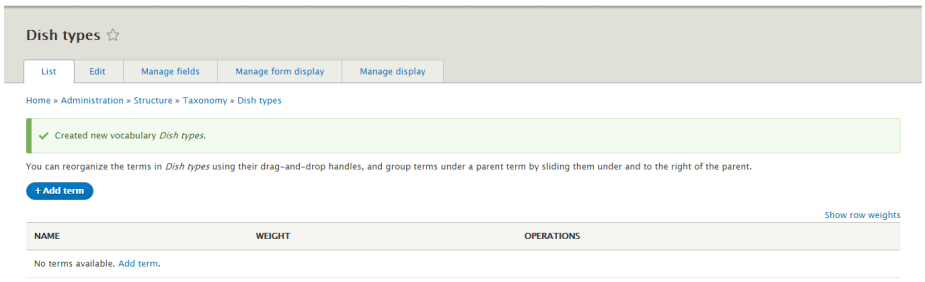



Figure 4.12: Adding a vocabulary terms

Click **Save**. In Figure 4.12 you can see the empty vocabulary. Next we will add some terms to the vocabulary. Click the **Add term** button and add the following terms:

- candy
- curry
- dessert
- pasta
- salad
- sandwiches
- sauces
- vegetarian
- vegan

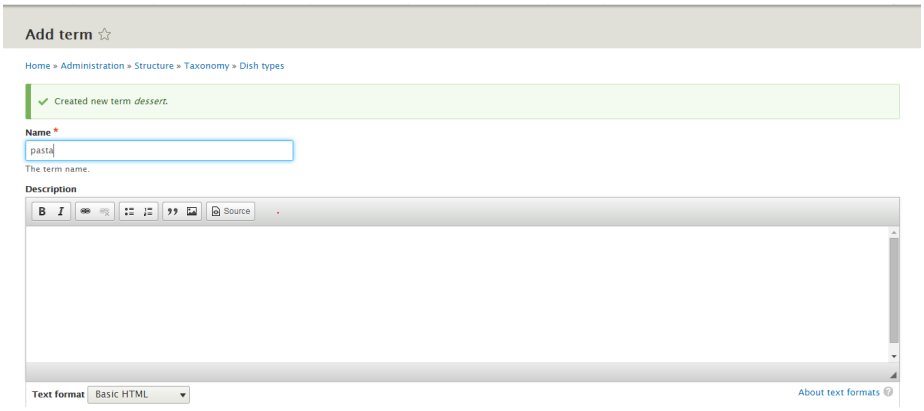



Figure 4.13: Adding a vocabulary term

To see an overview of the terms you have added go to **Structure** → **Taxonomy** and click the **List items** button next to your vocabulary. (Figure 4.14)

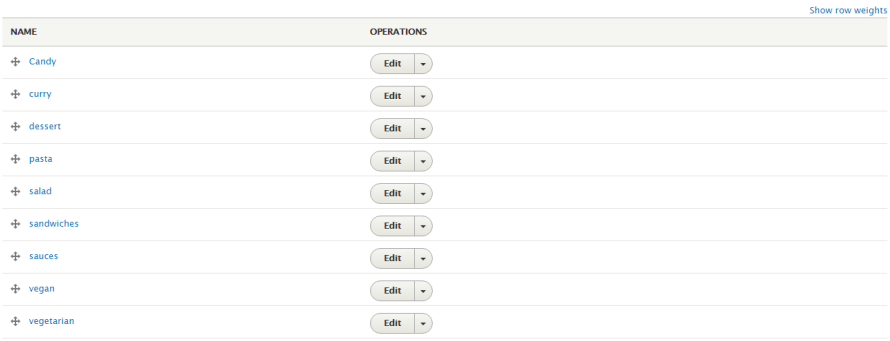



Figure 4.14: Vocabulary terms

### 4.2.4 Adding the Recipe content type

Since we are going to store recipes in our CMS we will need to add the **Recipe** content type. Go to: **Structure** → **Content types** → **Add content type**. Give it the following properties:

**Name:** Recipe

**Description** A recipe for cooking bugs!

At the bottom of the page you can see some settings for this content type. Explore the settings, the names are very descriptive so most of them should be clear without further explanation. These are general settings that apply to all instances of this content type. You are able to change these for each instance individually when you create them.

Click **Save and manage fields**.

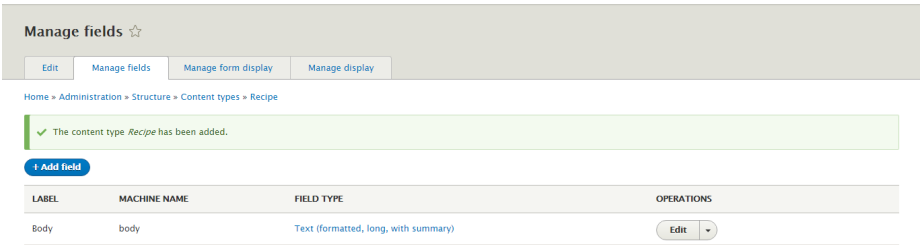



Figure 4.15: Manage recipe fields

Add the following fields to the **Recipe** content type:

**Name** (Text/plain, Maximum length = 255, number of values = 1)

**Ingredients** (Text/plain, Maximum length = 255, number of values = unlimited)

**Body** (already there)

**Plate image** (Image, number of values = 1)

**Estimated time** (Number(decimal), number of values = 1, Help text = Estimated time to cook the recipe in minutes).

**Type** (Taxonomy term, number of values = unlimited, reference method = default, Vocabularies: Dish types, Create referenced entities if they don't already exist).

In figure 4.16 you can see an overview of the fields.

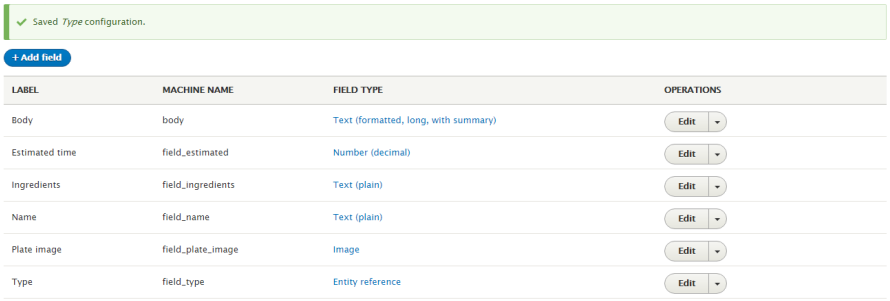



Figure 4.16: Recipe fields

## 4.3 review exercises

1. Log in to your **exploringdrupal8** site (created in the previous chapter). Add the search block to the **sidebar second** region and disable the **powered by Drupal** block.

2. Add a comment type **Answer** to your **exploringdrupal8** site. We will use the comment type to allow users to answer a question. The comment type has two fields: answer (a number between 0 and 100000) and motivation (a textual explanation describing how they got the answer).

3. Add a new content type **recipe** to your **exploringdrupal8** site. The new content type has the following fields: Title, PlateImage (Image), ingredients (list), description. Make sure the teaser only displays the Title and Image fields.

4. Add a vocabulary **Food types** to your **exploringdrupal8** site. Add the following terms to the vocabulary: Indian, Chinese, vegetarian, vegan.

# Chapter 5

# Creating content

## 5.1 Creating a basic page

In the previous chapter we learned how to create a content type. In this chapter we will learn how to create the content itself. An instance of a content type (a.k.a. a piece of content) is called a **Node** in Drupal. Creating content is easy, just go to **Content** → **Add content**. On this page you will see an overview of the different content types (Figure 5.1)


Figure 5.1: Add content, overview content types

We are going to add a basic page. This page will explain why it is good to eat bugs. Go to **Content** → **Add content** → **Basic page**. Use the following settings:

**Title** Why bugs

**Body** See file `chapter5/reasonstoeatbugs.txt` included in the course attachments.

On the right side you have several menu items. These allow you to change your posts properties. Under **Menu settings** add a menu link to the **Main navigation** (figure 5.2)

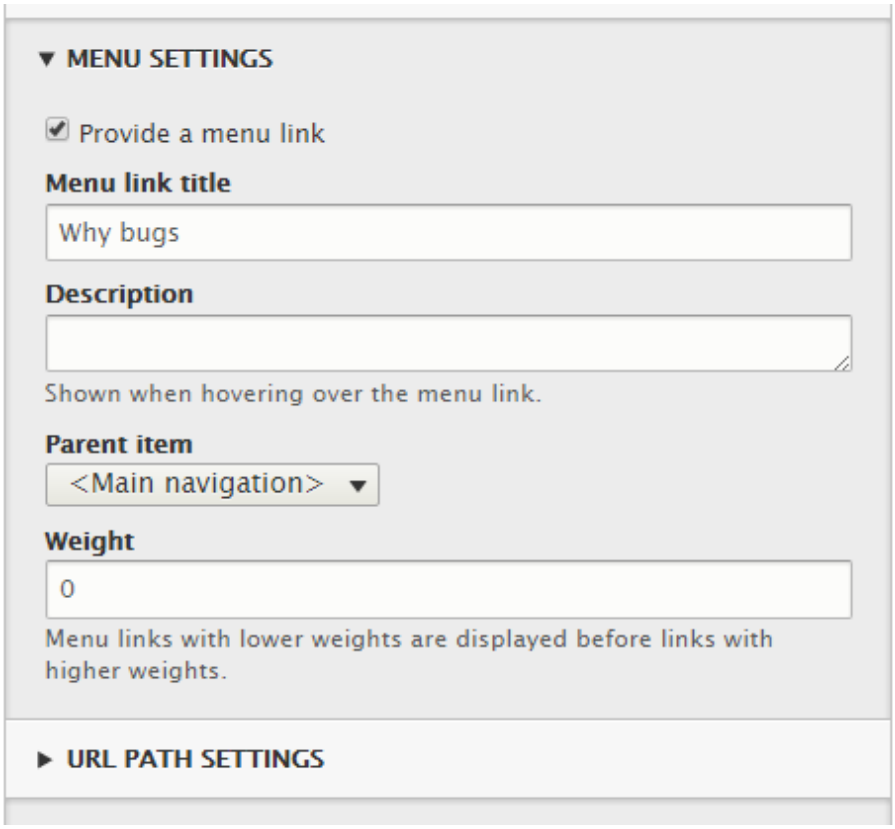



Figure 5.2: Basic page publishing options

Click **Save and publish**

## 5.2 Creating a recipe

Go to **Content** → **Add content** → **Recipe**. Figure 5.3 shows the editing page for our recipe. As you can see we have a **Name** and **Title** element. We don't need the title element. Remove it by going to: **Structure** → **Content**

**types → Recipe → Manage fields → Manage form display**. Move the edit fields like in figure 5.4

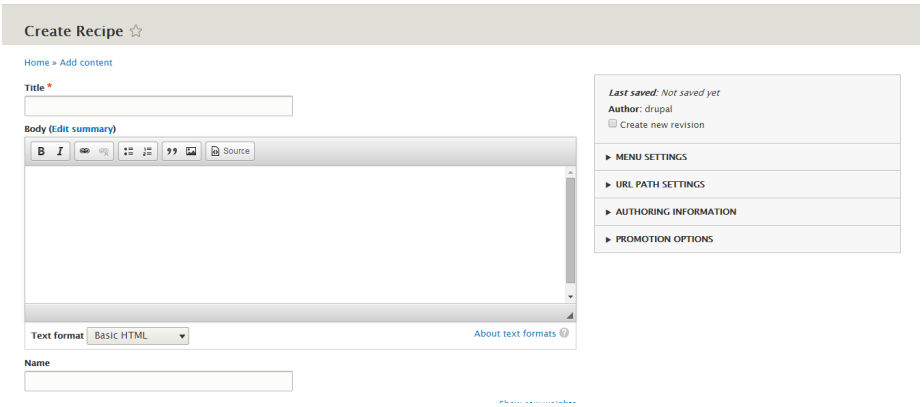



Figure 5.3: Create recipe form.

| FIELD | WIDGET |
|---|---|
| Name* | Textfield |
| Ingredients* | Textfield |
| Body | Text area with a summary |
| Plate image | Image |
| Estimated time | Text field |
| Type | Autocomplete |
| Promoted to front page | Check boxes/radio buttons |
| **Disabled** | |
| Sticky at top of lists | - Hidden - |
| Authored on | - Hidden - |
| Authored by | - Hidden - |
| URL alias | - Hidden - |
| Title | - Hidden - |
| Language | - Hidden - |

Figure 5.4: Manage recipe form display.

Now go back to the content creation page: **Content** → **Add content** → **Recipe**. Use the following information:

**Name** Dry Roasted Crickets

**Ingredients** 25  50 live crickets

**Body** See file `recipe_roasted_crickets.txt`

**Image** See file `roasted_crickets.jpg`

**Alternative text** Plate of bugs

**Estimated time** 10

**Type** snack

Click **Save and publish**. You should see a page like figure 5.5.

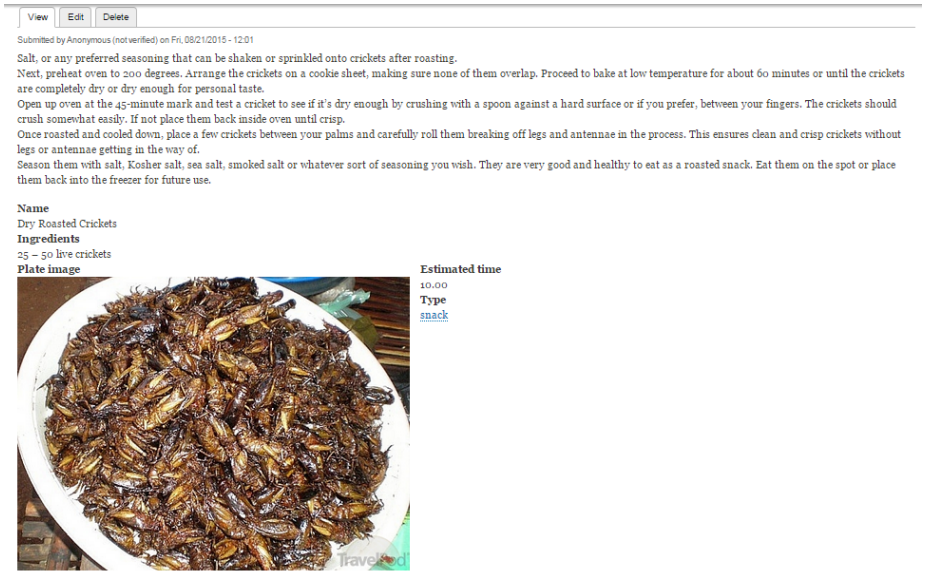



Figure 5.5: Roasted crickets recipe.

We would like to change the way that our recipe is displayed. To do this go to: **Structure** → **Content types** → **Recipe** → **Manage display**. Change it to look like figure 5.6

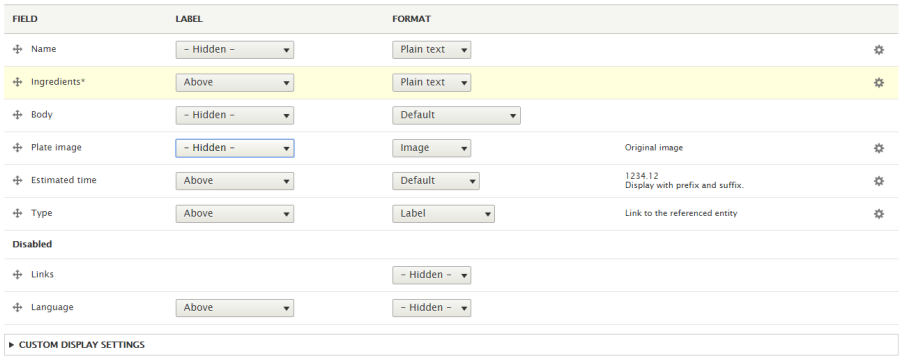



Figure 5.6: Manage recipe display

Click **Save**. The new recipe page should look like figure 5.7

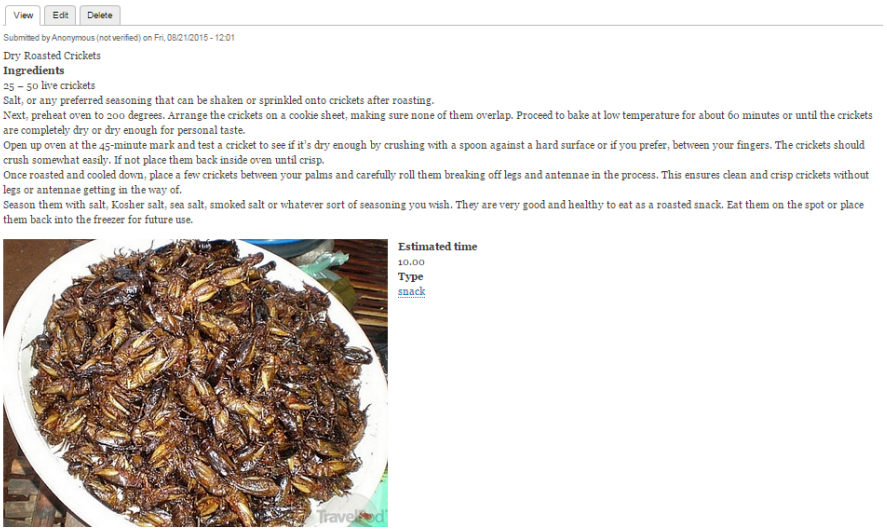



Figure 5.7: Recipe roasted crickets

To be honest, is looks a bit better than before but still not great. In the chapter about theming we will learn how to further change the layout by adding custom css.

## 5.3 Review exercises

1. Go to the homepage of the Biting Bugs site. You should see a summary of the recipe we added earlier in this chapter (Figure 5.8).

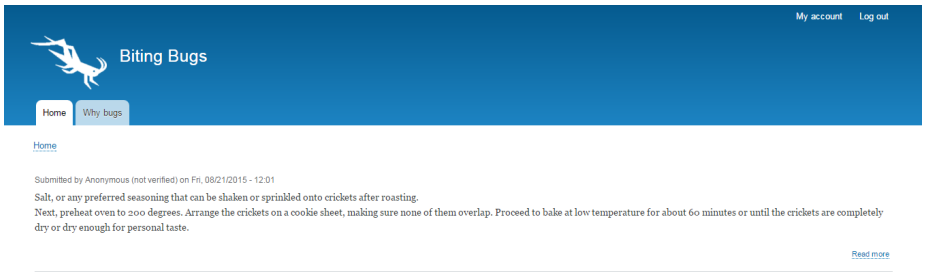



Figure 5.8: Recipe teaser

Change the display settings of the recipe content type teaser display mode so it looks like figure 5.9. Make sure that when you click the image you go to the full recipe.

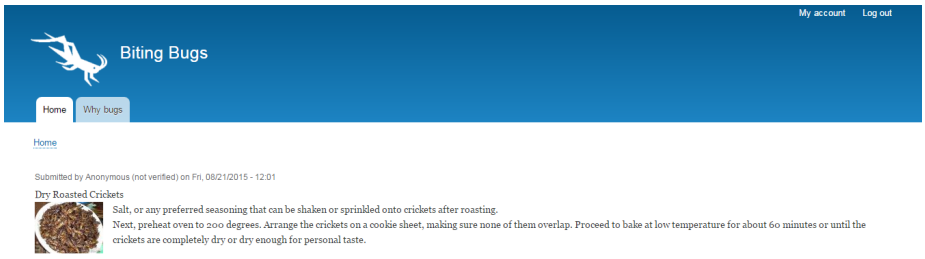



Figure 5.9: Recipe teaser updated.

HINT: You can edit the image size by clicking on the gear next to the image field.

2. Add three recipes to the bitingbugs site. You can find the recipes in the file `chapter5/recipes_with_bugs.txt`. Use the images: `cricket-flour.jpg`, `cricket_fried_rice.jpg` and `cricket_pad_thai.jpg`.

3. Logout of the bitingbugs site. On the homepage you can see a login form on the left side of the page. Make sure the login form does not appear on the homepage. (HINT: after removing the login form from the homepage you can always login by going to **site hostname**/user for example `bitingbugs.dd:8083/user`)

4. Change the display settings for the recipe teaser title so it links to the content.

5. Change the Recipe content type so the author information is not displayed.

6. Add a new basic page to the main menu. The basic page contains the information found in file `chapter5/edible_bugs.txt` and has the title **Edible bugs**. (Put the bug names in bold font.)

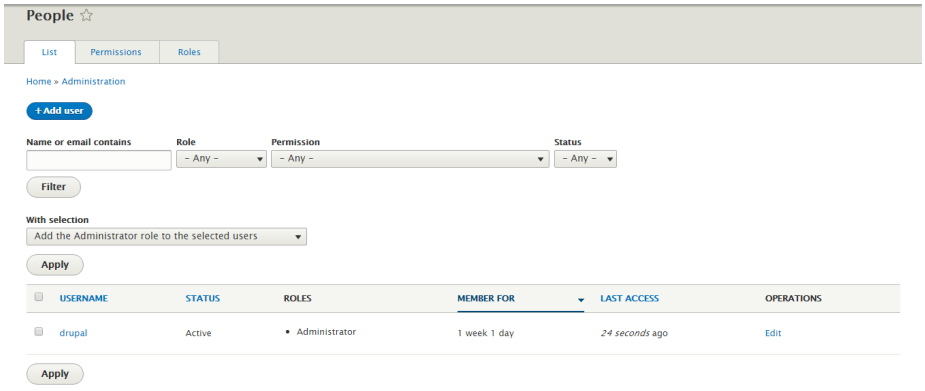
# Chapter 6

# Drupal Views

## 6.1 User management

Before digging into the Drupal Views module we are going to learn some things about user management. Out of the box Drupal provides powerful user management features. We can create users with different roles. A role defines a set of permissions. For example: a **moderator** role could have the permissions **Administer comment types and settings** and **Administer comments and comment settings**. To manage the user settings go to the **People** tab in the management menu (Figure 6.1).


The **People** page has three tabs: **List**, **Permissions** and **Roles** (Figure 6.1).



These three pages define the following functionality:

**List** Shows a list of all users, allows you to search trough the users and edit them.

**Permissions** This page lists all the site permissions vertically and user roles horizontally. For each role you can select its permissions by checking the check-boxes (6.1).

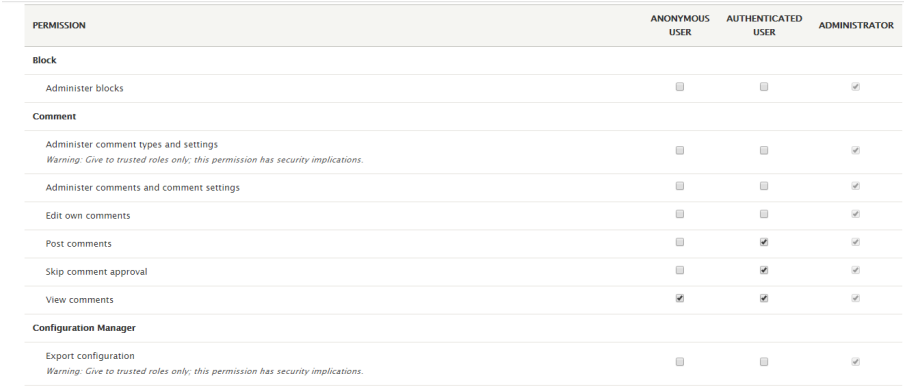



Figure 6.1: Permissions page.

**Roles** This page allows you to add, remove and edit roles. When you add a new role, it will appear on the permissions page.

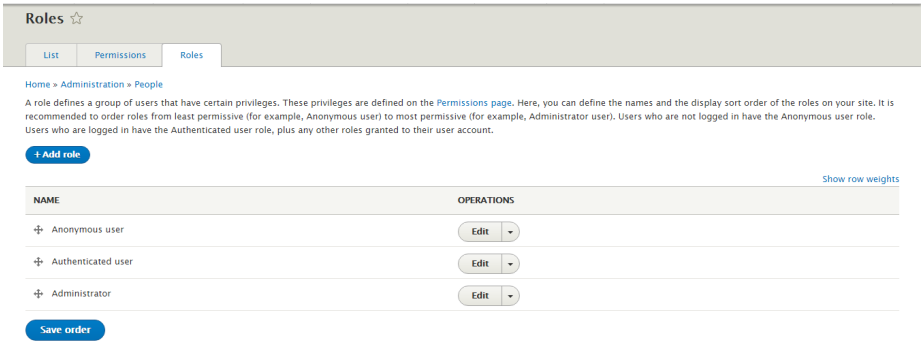



Figure 6.2: Roles page.

Next we are going to add a new role to our bitingbugs site. The role will be called **Chef**. A **Chef** can only add recipes to our site. Go to **People** → **Roles** → **Add role**. Give the new role the name **Chef** and click **Save** (Figure 6.3).


Figure 6.3: Added chef role.

To change the permissions for this role go to **People** → **Permissions**. Under the **Chef** role check the boxes next to: **Recipe: Create new content**, **Recipe: Delete own content**, **Recipe: Edit own content** and **Use the administration toolbar** (Figure 6.4).

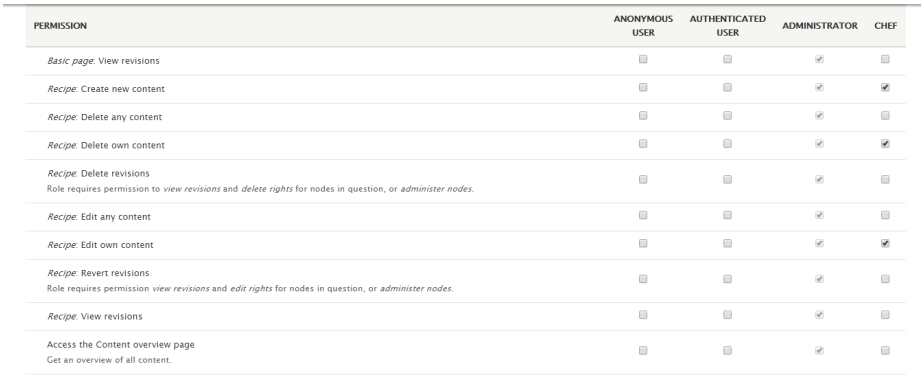



Figure 6.4: Edit role permissions.

Finally we will add a new user with the role **Chef**. Go to **People** → **Add user**, enter the following information:

**Email address:** chefjohn@bitingbugs.be

**Username:** Chefjohn

**Password:** ICanCook

**Status:** Active

**Roles: Authenticated user** and **Chef**

**Picture: chapter6/chefjohn.jpg**

**Site language:** English

**Contact settings:** Personal contact form

**Time zone:** Europe/Brussels

Click the **Create new account** button. By default Drupal redirects back to the **Create new account** page because you usually want to add multiple users. We will not add any more users for now. Go back to the **People** page, there you can see the new user we added. When we login with the chefjohn account we see the following menu:

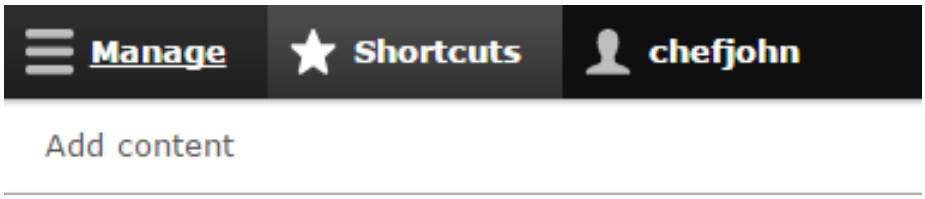



Figure 6.5: The menu for the **Chef** role.

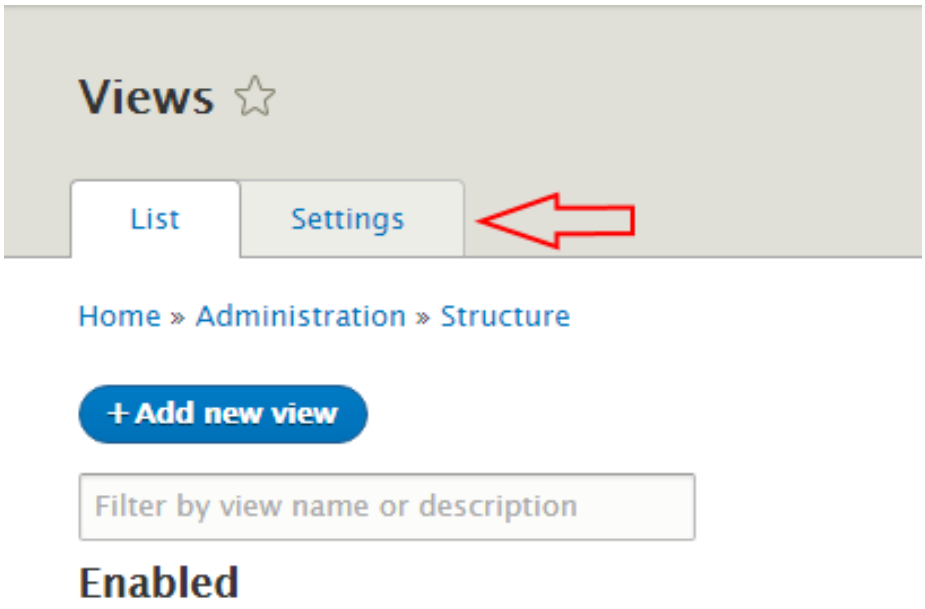
As you can see a **Chef** can only add recipes.

## 6.2 Views

### 6.2.1 Settings

Views is a query builder. It basically builds lists of content, and displays them in any format we choose. Views comes with a few standard preconfigured lists, e.g. a list to replace the standard front page, a list of recent comments, etc...

Views has a few settings that we like to turn on. On the **Structure** → **Views** page, click the **Settings** tab.




This will take you to the following page.

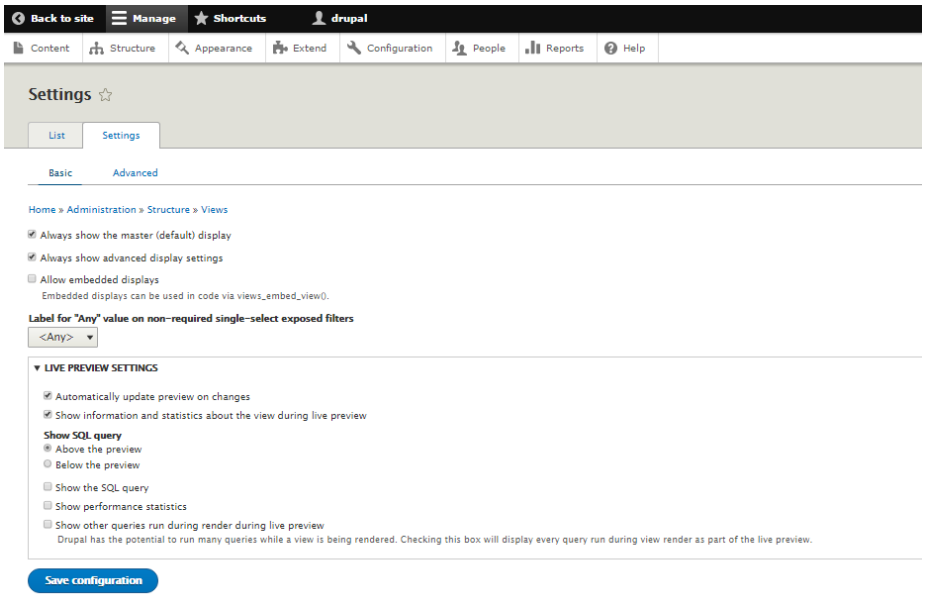



Figure 6.6: The views settings page.

Enable the **Always show the master display** and **Always show ad-**

**vanced display settings** settings.

Also enable the **Automatically update preview on changes** setting.

The **Show information and statistics** option and the **SQL query options** are useful when there is a problem and you are debugging what is wrong, you can turn this on as well.

### 6.2.2 Creating a view

To create a new view go to **Structure** → **Views** → **Add new view**.

Now you can provide a View name, and choose what you want to display (content, taxonomy, files, users...).

Besides that, you can also choose one or two displays to start with. If you want your view to be displayed as a page, select the **Create a page** option. You will have to provide a Page title and Path. All these settings can be altered afterwards.

You can also choose to display the view in a block by selecting Create a block. In this case you will need to give it a **Block title**. When you are done, click **Save and edit** (Figure **??**).

## 6.3 Review exercises

1. Add a new role **Content manager**. A content manager can create, delete and edit all content but nothing else.

2. Create a new content manager named **managerpaul**.

# Chapter 7

# Collaborating on a Drupal site

Throwing your codebase into Git or an other versioning system will not be enough to enable you to collaborate on your Drupal site. Since Drupal uses a database to store its contents this database should be shared as well. In this course we recommend using the OpenShift cloud for hosting your site. OpenShift is a platform as a service product from Red Hat. The software that runs the service is open-sourced under the name OpenShift Origin, and is available on GitHub. Developers can use Git to deploy web applications in different languages on the platform. A version for cloud computing is named OpenShift Enterprise. OpenShift also supports binary programs that are web applications, as long as they can run on Red Hat Enterprise Linux. This allows the use of arbitrary languages and frameworks. OpenShift takes care of maintaining the services underlying the application and scaling the application as needed.

## 7.1 Creating an openshift account

- First go to `https://www.openshift.com/`
- Click on **Sign Up**
- Fill out your information
- Click **Sign Up**
- Verify your account
- Accept the terms of service
- Click **Create your fist application now**

OpenShift gives you the option to create an instant Drupal 7 app. Since we are using Drupal 8 we will not use this option. We are going to create a PHP 5.4 application. This is a Linux server running a PHP web hosting environment. Click the PHP 5.4 application link and enter the following information:

**Public url** drupalsite-[firstname][lastname].rhcloud.com

**Source code** leave empty

**Scaling** No scaling

**Region** aws-us-east-1

Click the **Create application** button.

When OpenShift asks if you will be changing the code for the application, click **Not now, continue**. You will see a page like figure 7.1.

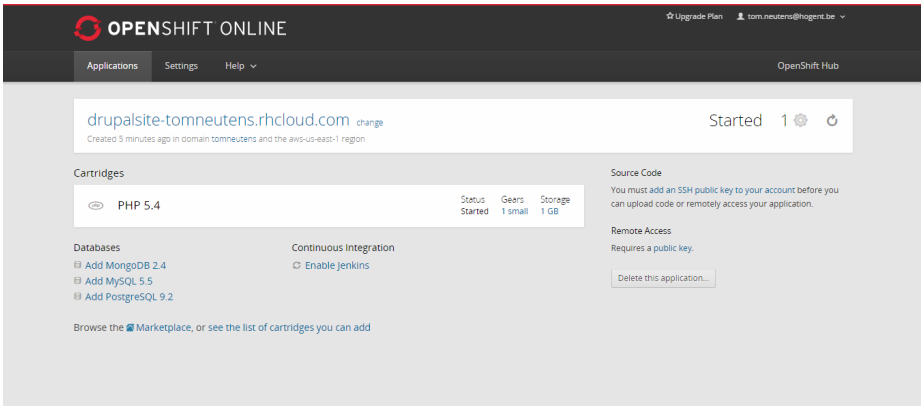



Figure 7.1: OpenShift application overview page.

Since we are going to use a MySql database, click the **Add MySQL 5.5** link. On the next page click **Add Cartridge**. Make note of the database credentials, it's a good idea to take a screenshot! After adding the MySQL cartridge you should add the PHPMyAdmin cartridge to manage the database.

## 7.2 Installing the OpenShift client tools

To be able to manage your OpenShift site you have to install some tools. The following link (`https://developers.openshift.com/en/managing-client-tools.html`) gives step by step instructions on how to install them.

## 7.3 Putting your site on openshift

To start we will get a local copy of our application repository through git. Create the folder `C:/openshift_repos/bitingbugs_example` on your local machine. Navigate to the new folder through Windows explorer. Right click somewhere on the window and select **Git bash** (figure 7.2).

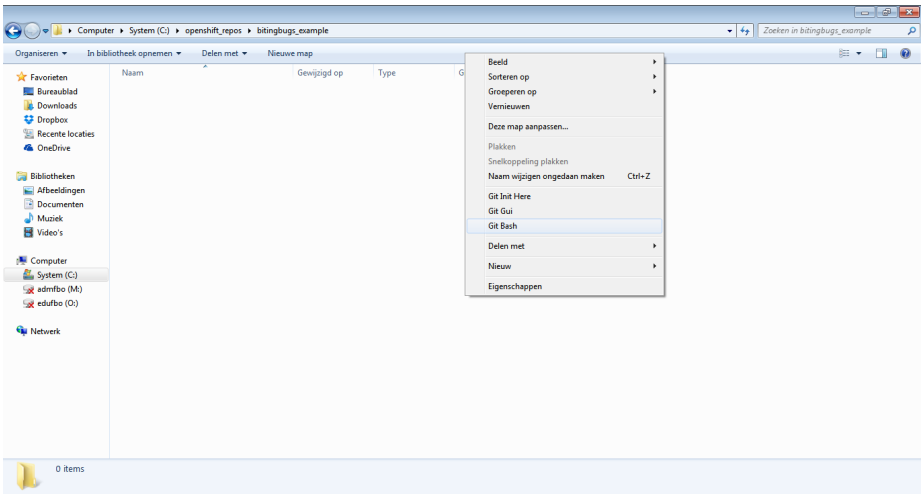



Figure 7.2: Open the Git bash shell.

We are going to clone our OpenShift repository to our local machine. If u followed the steps in the previous section correctly, you should find the url to your repository on your OpenShift application page (Figure 7.3).

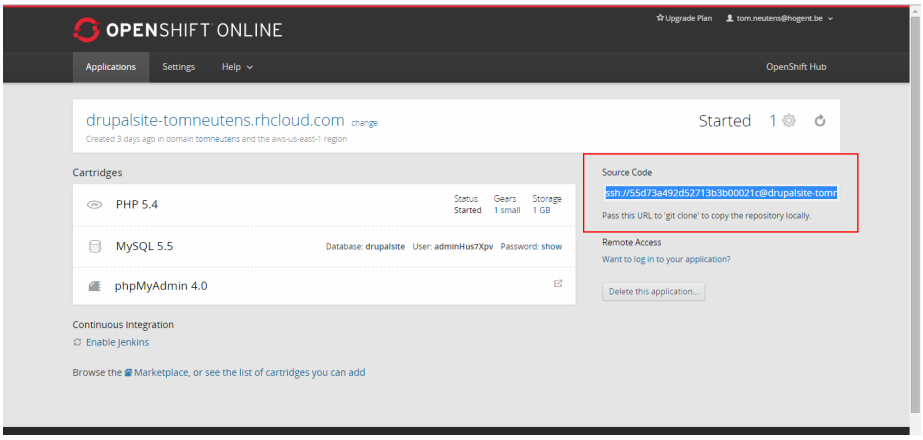



Figure 7.3: Open the Git bash shell.

Go back to the Git bash shell and type the following command **git clone <git-url >**. This will create a local copy of your OpenShift code. For now this code is very limited. If you look in the **drupalsite** folder you will only see one file, the `index.php`. In the next step we are going to replace the `index.php` file by our Drupal site code. Go to the folder where you stored the code for the bitingbugs example (figure 7.4) and copy it to your new repository (figure 7.5).

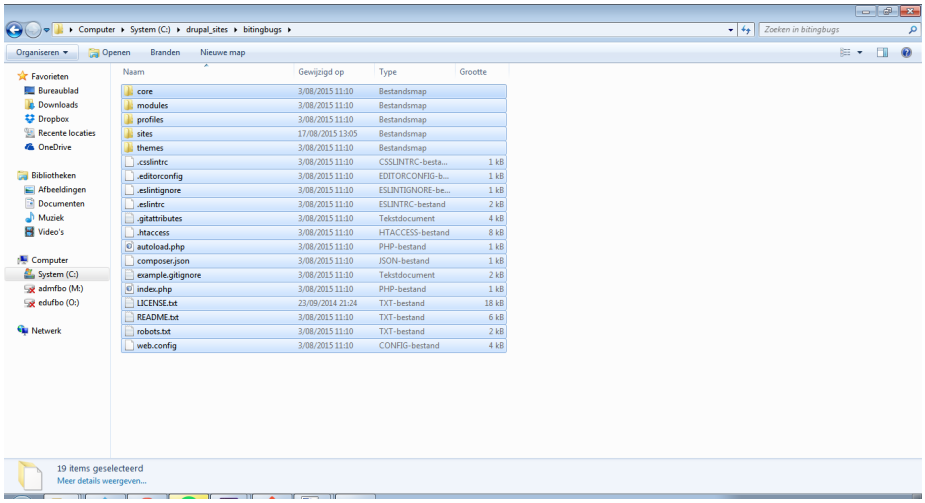



Figure 7.4: The local codebase, this is the code managed by the Acquia Dev desktop.

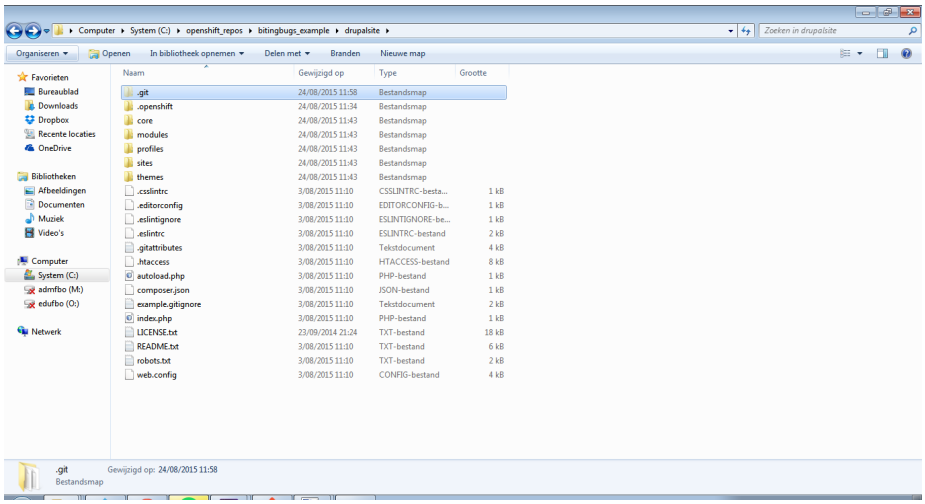



Figure 7.5: The location of your OpenShift local git repository.

To add your Drupal site into the git versioning system go back to the `drupalsite` folder and execute the command **git add ***, this might take a while (figure 7.6). You can use the **git status** command to see all the files that where added.

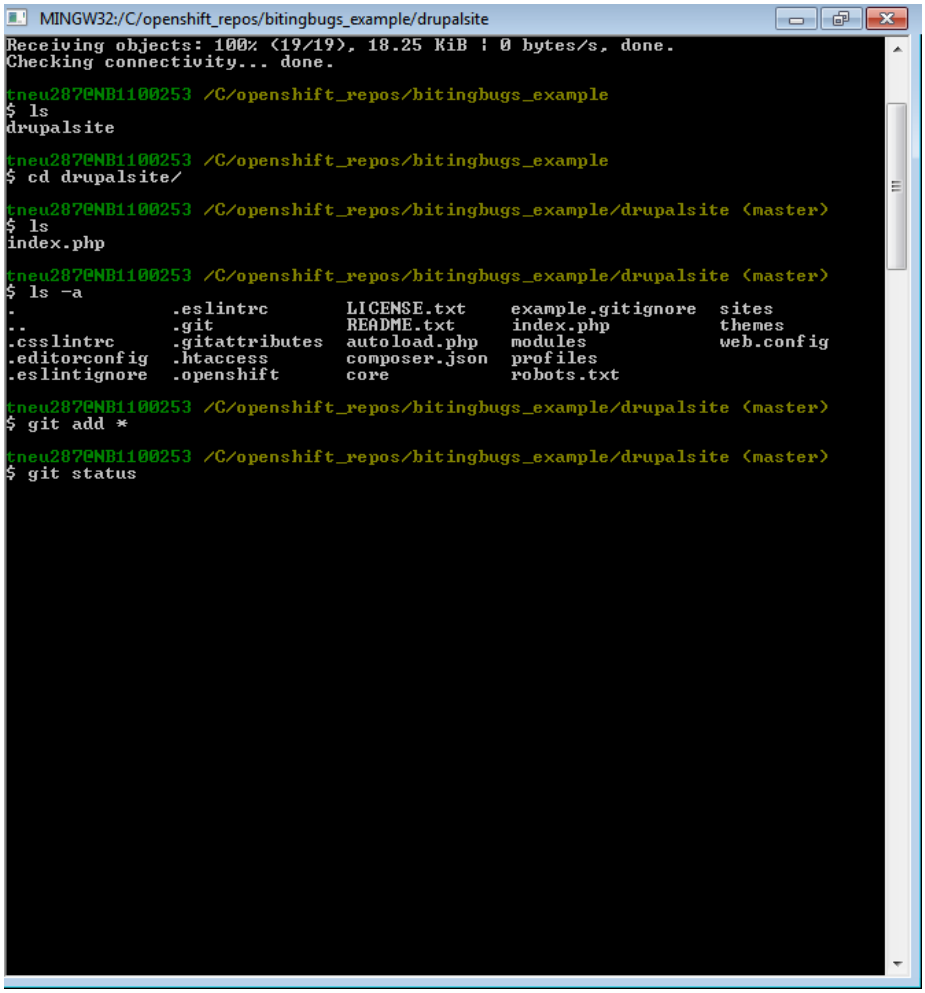



Figure 7.6: Git add command.

After running the **git status** you should see the following result:

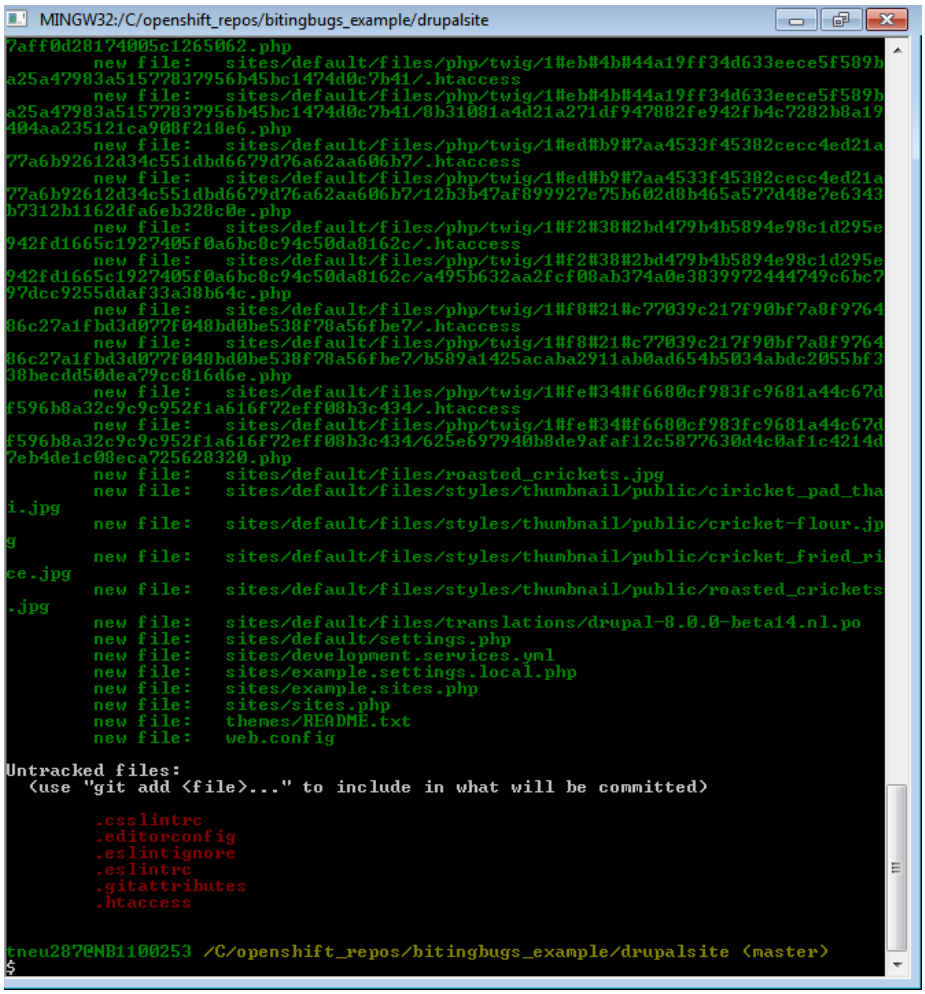



Figure 7.7: Git status command.

As you can see some files are not added. To make sure routing works correctly you should add the `.htaccess` file. (**git add .htaccess**). Next we commit the new files to the local repository. Use **git commit -m "initial commit, adding all files"**. The **-m** parameter (commit message) is not mandatory but it is a good practice to always add it!

## 7.4 Migrating your database to OpenShift

Our code has been uploaded to OpenShift but our OpenShift database is still empty. First we will make a dump of our local database, afterwards we will upload the data to our OpenShift database. To create the dump file go to the

phpMyAdmin page of your local site. You can do this by clicking the link in the Acquia Dev desktop. Go to the **Export** tab and click the **Start** button. Next go to your OpenShift account and go to your application page. Click the link to the phpMyAdmin page (figure 7.8). And enter your credentials (found on your application page).

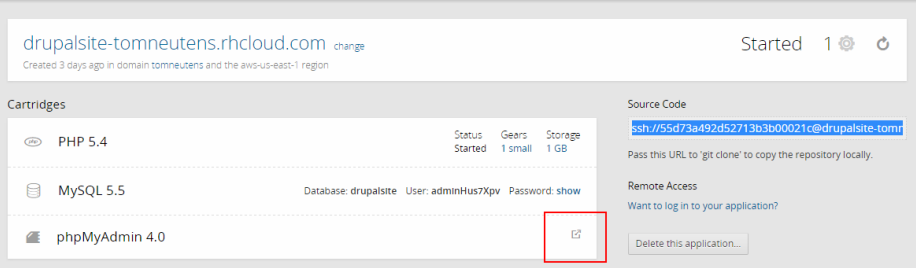



Figure 7.8: OpenShift phpMyAdmin link.

When you are logged into your OpenShift phpMyAdmin page, first select the drupalsite database in the left menu. Next click the **import** tab. On the import page, select your local dump file and click the **Start** button.

# Chapter 8

# Drupal commerce

## 8.1 Installing the Commerce module(s)

Drupal commerce is a set of Drupal modules which add web store functionality to our site. The project is maintained by the commerce guys (`https://commerceguys.com/`), it is free and opens source (`https://github.com/commerceguys`). Since no official version of Drupal commerce has been released for Drupal 8, we are going to use the development version. This could lead to some unexpected problems but as you know, developers like lining on the edge. The installation instructions can be found here: `https://github.com/commerceguys/commerce`. To install the modules we are going to use drush, which stands for Drupal SHell. Drush is a command line tool which makes some tasks a lot easier. To open a Drush console you can click the console window button in the Acquia Dev Desktop (Figure 8.1).

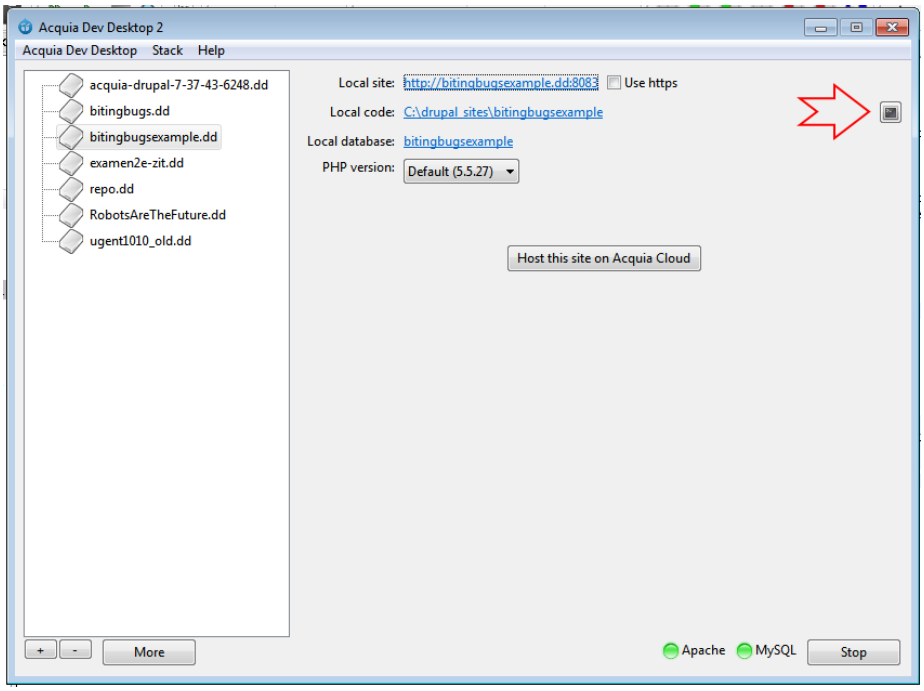



Figure 8.1: Launch drush console button.

We will start by installing the **composer_manager** module. Execute the following commands:

```
drush dl composer_manager  #downloads composer_manager module
drush en composer_manager -y      #enables composer_manager module
```

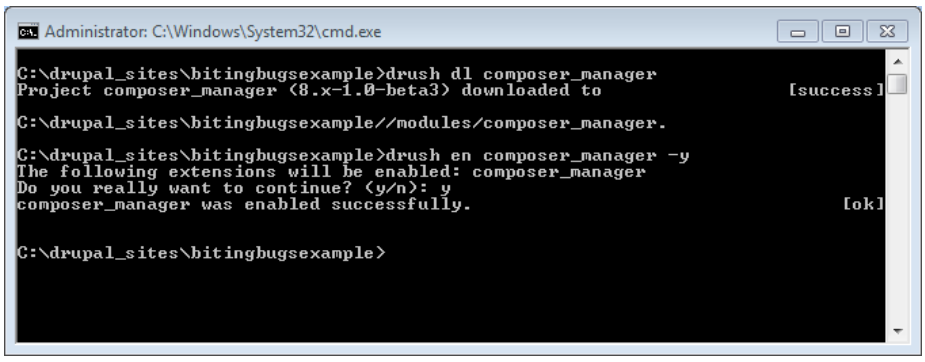



Figure 8.2: Downloading and enabling the composer_manager module.

The next step is initializing composer_manager:

```
drush composer-manager-init
```

The next step is installing the commerce module:

```
drush dl commerce  #this generates an error!
```

The command above downloads the commerce module and tries to load the dependencies. The download is successful but the loading fails. This is because our drush installation can not find the `composer.phar` file. To solve this we are going to use absolute path values. On my computer the command is the following (you should change the path to match your Acquia Dev Desktop installation directory).

```
cd core #change to the core folder
"C:\Program Files (x86)\DevDesktop\php5_6\php"
"C:\Program Files (x86)\DevDesktop\drush\composer.phar" drupal-update
#perform a drupal update
```

Now we will enable the commerce module. Go to the bitingbugs site and go to: **Extend**. On the extend page there is a search box, search for *commerce*. Select the **Commerce** module and click the **Install** button.

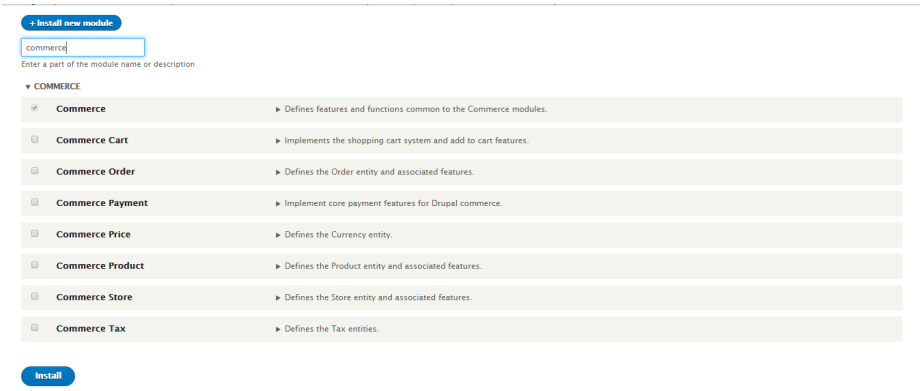



Figure 8.3: Enable Drupal commerce module.

Do the same for the other commerce modules. You should enable them one by one because installing them all at once could lead to problems. After enabling the commerce module you should see a new link in the admin menu (Figure 8.4).


Figure 8.4: Commerce admin menu item.

## 8.2 Adding a webstore to bitingbugs

# Chapter 9

# Introduction to php

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. As of January 2013, PHP was installed on more than 240 million websites (39 percent of those sampled) and 2.1 million web servers. Originally created by Rasmus Lerdorf in 1994, the reference implementation of PHP (powered by the Zend Engine) is now produced by The PHP Group. While PHP originally stood for Personal Home Page, it now stands for PHP: Hypertext Preprocessor, which is a recursive backronym.

PHP code can be simply mixed with HTML code, or it can be used in combination with various templating engines and web frameworks. PHP code is usually processed by a PHP interpreter, which is usually implemented as a web server's native module or a Common Gateway Interface (CGI) executable. After the PHP code is interpreted and executed, the web server sends the resulting output to its client, usually in the form of a part of the generated web page; for example, PHP code can generate a web page's HTML code, an image, or some other data. PHP has also evolved to include a command-line interface (CLI) capability and can be used in standalone graphical applications.

The standard PHP interpreter, powered by the Zend Engine, is free software released under the PHP License. PHP has been widely ported and can be deployed on most web servers on almost every operating system and platform, free of charge.

Despite its popularity, no written specification or standard existed for the PHP language until 2014, leaving the canonical PHP interpreter as a de facto standard. Since 2014, there is ongoing work on creating a formal PHP specification.

## 9.1 PHP introduction

Since we assume you are familiar with other programming languages, we will only go over the basics of PHP. For this we refer to the powerpoint presentation `Intro_php.pptx`.

# Chapter 10

# Drupal theming

In this chapter we are going to learn how to change the look of our site. First we will learn how to change the basic theme settings. Next you will learn how to install a custom theme. The last part of this chapter explains how to extend a theme and add your own html and css.

## 10.1 Theme basics

### 10.1.1 Changing the theme settings

To view the settings of your current theme (Bartik) go to **Appearance** → **Settings**. On his page you can see a tab for each of the enabled themes (Figure 10.1).

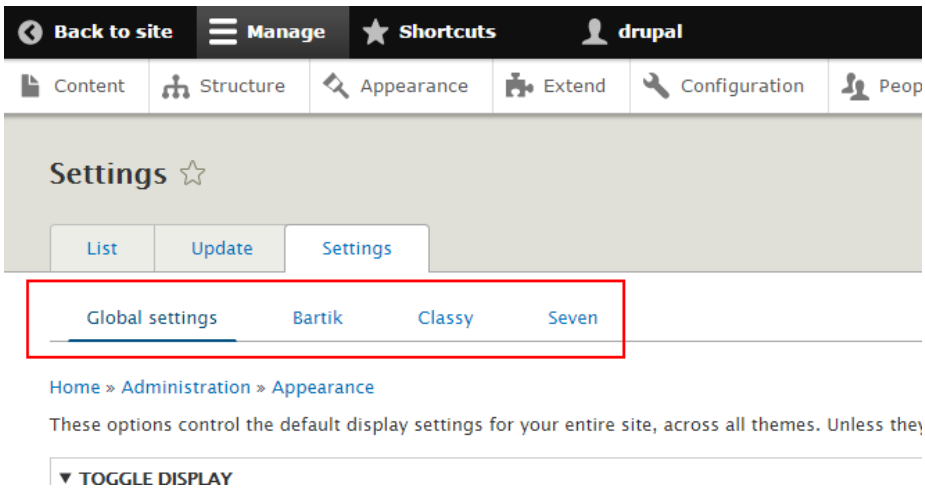



Figure 10.1: Theme settings, overview enabled themes.

Some themes will come with more settings, such as colour options. Compare the themes you have enabled. Many theme specific settings are very similar to the global settings. When you change settings per theme, they will override the Global settings for that theme. Some of the more interesting themes will offer much more functionality. The default Bartik theme implements functionality from the Colour module, play around with them a little. The Colour module can, but does not have to be implemented in a theme, so not every theme will offer this functionality.

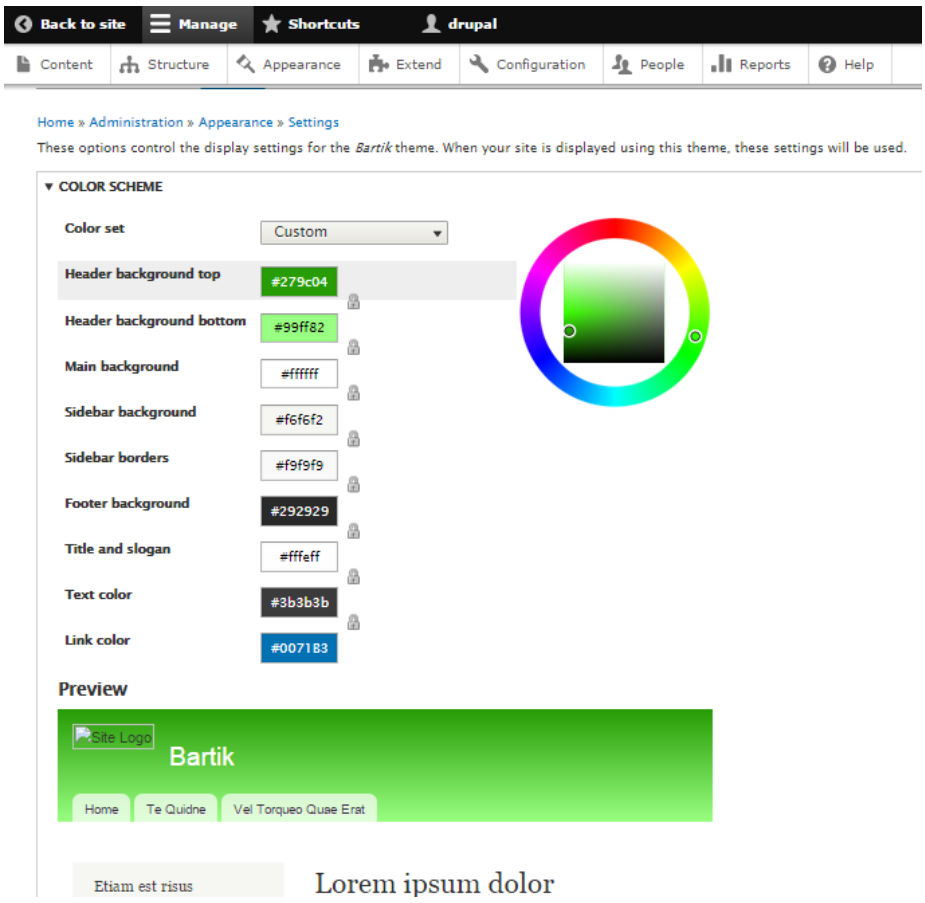



Figure 10.2: Changing the Bartik color settings.

Notice in figure 10.2 the logo image is not displayed. This is because we are using a custom logo image.

### 10.1.2 Installing a new theme

In Drupal there are several ways to install a theme. The fist option is to go to **Appearance** → **Install new theme**.

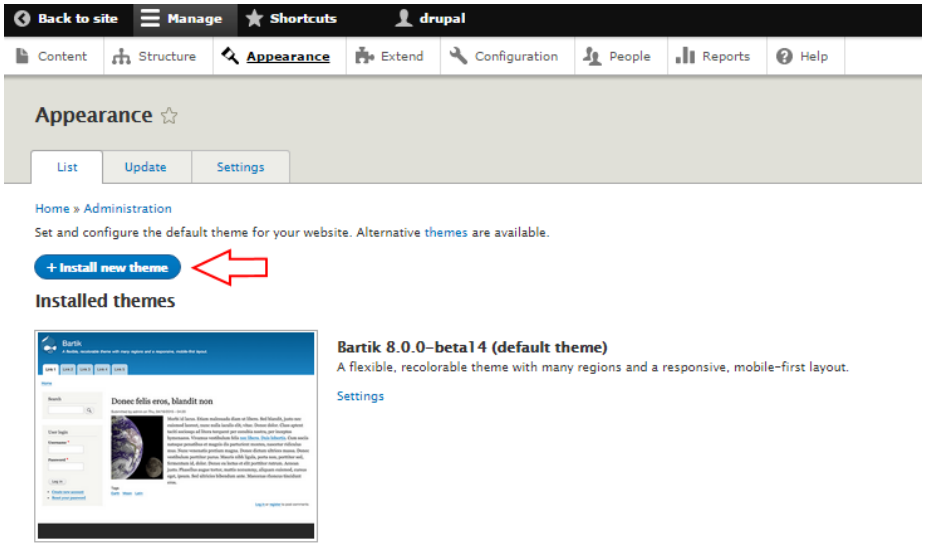



Figure 10.3: Click the button to install a new theme.

This will take you to a page where you can either install from a URL, or upload an archive from your local computer.

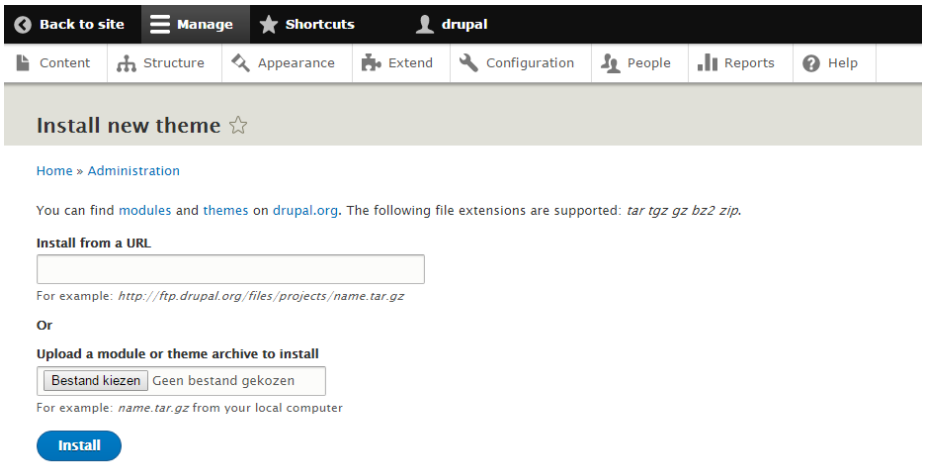



Figure 10.4: On this page you can choose to install a theme by copying the url from the location where the theme is hosted or uploading the theme from your local computer.

Either way, you will have to visit the theme project page at `https://drupal.org/project/theme_name` and find the file URL for the archive, or just download it, which takes us to option number two.

The second way to install a new theme downloading it, unpacking it and placing it in the appropriate folder, which is `/themes` (Figure 10.5).

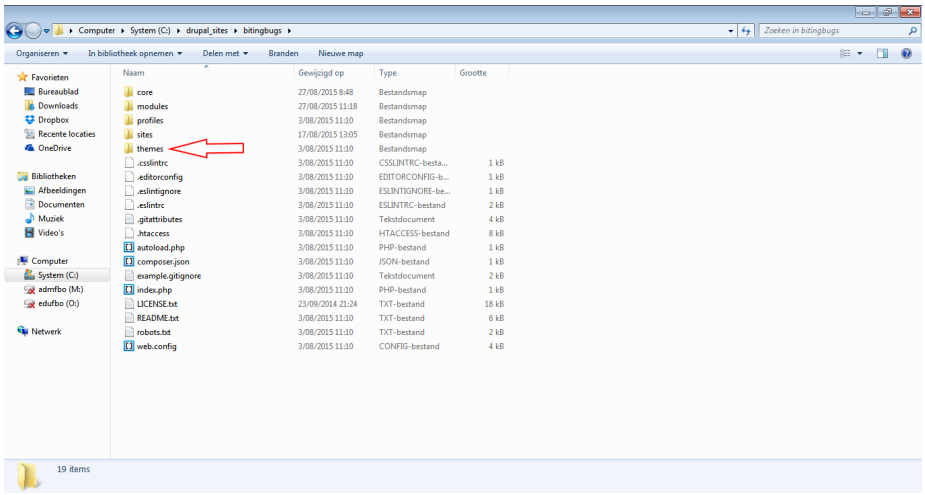



Figure 10.5: Theme install location.

The third and quickest way to install a theme is through the Drush command line tool. Using Drush we have to type the following two commands:

```
drush dl theme_name
drush en theme_name
```

The theme_name, should be the machine name. This can be found in the URL of the project page on drupal.org, it is the word after project/. For example in `www.drupal.org/project/bartik` the machine name is **bartik**.

Drupal 8 has all core code and themes under a directory named /core. For contributed and custom themes drupal uses the /themes folder that lives on the same level as the /core folder. For people having experience with drupal 7 themes, these are the 2 most apparent changes when we look at a theme folder:

- The *.infofile changes to *.info.yml.
- The *.tpl.phpfiles change to *.html.twig.

The README.txt file in the `/themes` directory says the following:

*"Place downloaded and custom themes that modify your site's appearance in this directory to ensure clean separation from Drupal core and to facilitate safe, self-contained code updates. Contributed themes from the Drupal community may be downloaded at `http://drupal.org/project/themes`. It is safe to organize themes into subdirectories and is recommended to use Drupal's subtheme functionality to ensure easy maintenance and upgrades. In multisite configuration, themes found in this directory are available to all sites. In addition to this directory, shared common themes may also be kept in the `sites/all/themes` directory and will take precedence over themes in this directory. Alternatively, the `sites/your_site_name/themes` directory pattern may be used to restrict themes to a specific site instance. Refer to the "Appearance" section of the README.txt in the Drupal root directory for further information on theming."*

## 10.2 Creating your own theme

To create a new theme navigate to your site `/themes` folder. For me this is the `C:\drupal_sites\bitingbugs\themes` folder. There we will create a new folder for our theme named **my_new_theme**. Inside the new folder we need a file to describe our theme. Drupal requires this file to be named: **[theme name].info.yml**. So in our case we create a file named **my_new_theme.info.yml**.

A `.yml` file is used for the YAML file type. YAML stands for YAML Ain't Markup Language. It is called this way because it strives to be as human readable as possible. Figure 10.6 shows an example of a YAML file. As you can see YAML files look very simple because they use whitespace to structure the content.

```yaml
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    given:   Dorothy
    family:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     100.27
      quantity:  1
```

Figure 10.6: An example of a basic YAML file.

The `.info.yml` file has four required keys (figure 10.7):

**Name:** The human readable name will appear on the Appearance page, where you can activate your theme.

**Type:** The type key indicates the type of extension, e.g. module, theme or profile. For themes this should always be set to "theme".

**Description:** The description is also displayed on the Appearance page.

**Core:** The core key specifies the version of Drupal core that your theme is compatible with.

```yaml
name: My new drupal 8 theme
type: theme
description: 'A subtheme of classy'
core: 8.x

```

Figure 10.7: Required info file keys.

Next to the required keys you can also add some other properties:

**Package:** The package key allows you to group themes together on the Appearance page.

**Base theme:** : The theme can inherit the resources from another theme by defining it as a base theme.

**Version:** For modules hosted on `drupal.org`, the version number will be filled in by the packaging script. You should not specify it manually, but leave out the version line entirely.

**Screenshot:** With the screenshot key you define a screenshot that is shown on the Appearance page. If you do not define this key then Drupal will look for a file named `screenshot.png` in the theme folder to display.

**Libraries:** The libraries key can be used to add asset libraries which can contain both CSS and JS assets to all pages where the theme is active.

**Stylesheetremove:** The stylesheetsremove key removes a link to a stylesheet added by another theme or module. The full path to CSS file should be provided (instead of just the filename), to accommodate cases where more than one file with the same name exists. In cases where a Drupal core asset is being removed (for example, a CSS file in jQuery UI) the full file path is needed. In cases where the file is part of a library that belongs to a module or theme, a token can be used. Note that when using the token it needs to be quoted because @ is a reserved indicator in YAML.

**Regions:** Regions are declared as children of the regions key. You are required to have a content region.

Although you only need a `.info.yml` file with the required keys to have a new Drupal theme, it will not be of much use. To have a basic functioning theme you should have the following `.info.yml` file:

```yaml
name: My new drupal 8 theme
type: theme
base theme: calssy
description: 'A subtheme of classy'
package: Custom
core: 8.x
libraries:
  - d8/global-styling
stylesheets-remove:
  - core/assets/vendor/normalize-css/normalize.css
regions:
  header: Header
  content: Content
  sidebar_first: 'Sidebar first'
  footer: Footer

```

Figure 10.8: A basic info.yml file.

Aside from the `.info.yml` file you should also have the `html.html.twig` and `page.html.twig` template files. A template file defines the structure of a certain part of your Drupal site. They contain html and some logic. However the logic in these files should be limited and is reserved for the **[theme name].theme** file. You will learn more about template files later in this chapter.

## 10.3 Subtheming

When we create a subtheme we take an existing theme and use it as a template for our own. The subtheme extends the base theme, this means that we add extra functionality on top of the existing theme. You might be wondering why we don't just modify the base theme. The advantage of extending the base theme is that when it gets updated, our modifications don't get overridden.

### 10.3.1 How to create a subtheme

Creating a sub theme is pretty much like creating a theme. The main difference is you (also) need to declare a parent theme in the *.info.yml file, like so:

```
1  name: My new drupal 8 theme
2  type: theme
3  base theme: calssy
4  description: 'A subtheme of classy'
5  package: Custom
6  core: 8.x
7  libraries:
8    - d8/global-styling
9  stylesheets-remove:
10   - core/assets/vendor/normalize-css/normalize.css
11 regions:
12   header: Header
13   content: Content
14   sidebar_first: 'Sidebar first'
15   footer: Footer
```

Figure 10.9: Subtheme declaration in the info.yml file.

If the base theme has regions and/or features defined in the *.info.yml file which you want to keep, youll need to copy those to the sub theme as well. Declaring only your own, will override any existing. Some contributed base themes require a little more to set up, make sure you refer to each themes README.txt file for full instructions on how to begin using it, as each is different. As usual with Drupal, there are several ways of doing things. Many of the more popular themes provide Drush integration to make it even easier. So always check the documentation.

When you create a sub theme, it will inherit everything from CSS to the screenshot from the base theme, with the exception of the regions, features and settings defined in the parents *.info.yml file. Template files will just work, theres no need to copy them or create your own, unless you want to override them. In that case, youll have to copy the right *.html.twig file in your themes templates folder. Functions from the parents .theme file will just work, and you can extend/manipulate them further in your own. Since these functions should be prefixed with your theme name, you can use exactly the same hooks to further manipulate data or output.

### 10.3.2 Creating a subtheme in practice

In this section we will create our own subtheme for the bitingbugs example. First create a folder for our new theme in the `/themes` folder (Figure 10.10).

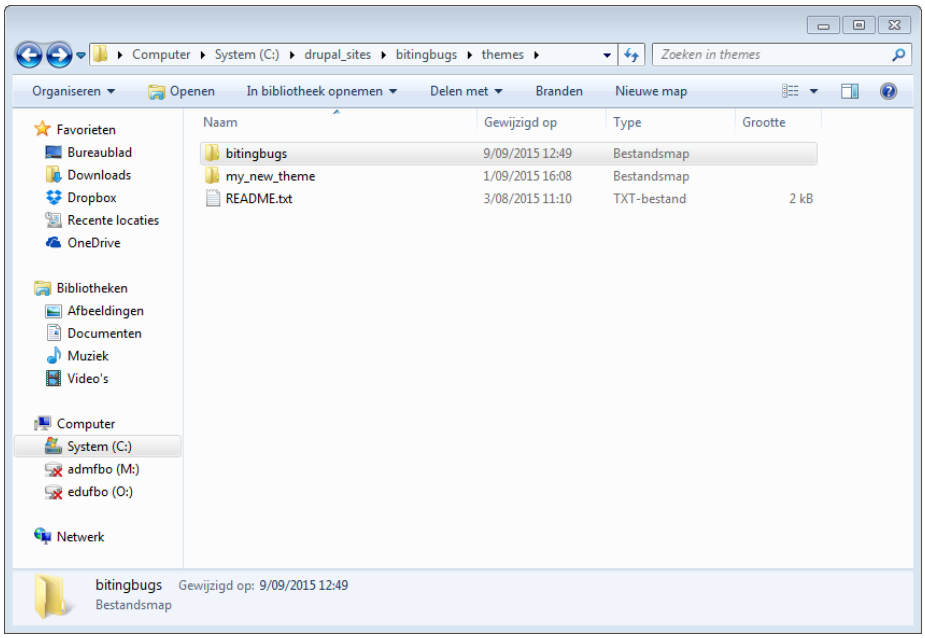



Figure 10.10: The folder for our new subtheme.

Next we will add our `bitingbugs.info.yml` file to our new theme folder, the file name should have the same name as your theme. This file provides metadata about your theme to Drupal. This is similar to how modules and installation profiles are being defined, and as such it is important to set the 'type' key in the *.info.yml file to 'theme' in order to differentiate it (Figure 10.11).

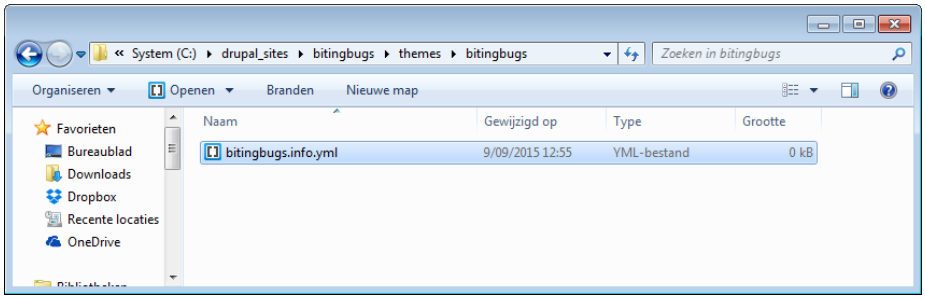



Figure 10.11: Bitingbugs theme info file.

Next we will add some content to our `bitingbugs.info.yml` file (Figure 10.12).

```
name: bitingbugs
type: theme
base theme: classy
description: 'A sub theme for our bitingbugs site based on classy'
package: Custom
core: 8.x
libraries:
  - bitingbugs/global-styling
stylesheets-remove:
  -core/assets/vendor/normalize-css/normalize.css
```

Figure 10.12: .info.yml file content.

If you have done the previous steps correctly, you should see your theme on the theme settings page of your Drupal site under **Uninstalled themes**.

### Adding regions to a sub theme

Adding regions to a theme requires:

- Adding region meta-data to your `.info.yml` file.
- Editing your `page.html.twig` file and printing the new regions.

### Adding region meta-data to your .info.yml file

Start by declaring any new regions in your bitingbugs.info.yml file. Regions are declared as children of the regions key like so:

```
regions:
  header: Header
  primary_menu: 'Primary menu'
  content: Content
  sidebar_first: 'Sidebar first'
  footer: Footer
```

Figure 10.13: .info.yml regions.

Region keys can contain: letters, numbers and underscores. Keys should begin with a letter. When referencing regions the key from your bitingbugs.info.yml file will be used as the machine readable name of the region, and is how you will reference the region in code. The human readable name of the region is what will be used in the user interface for anyone administering regions.

### Adding regions to your templates

In order for regions to display any content placed into them you'll need to make sure your new regions are also added to your page.html.twig file. Regions will be represented as Twig variables whose name corresponds with the key used in your bitingbugs.info.yml file with the string page. prepended.
**Example:**

```
header: 'Header'
```

**Becomes:**

```
{{ page.header }}
```

### Default regions

See the page.html.twig documentation for a list of default regions.

- page.header: Items for the header region.
- page.primary_menu: Items for the primary menu region.
- page.secondary_menu: Items for the secondary menu region.
- page highlighted: Items for the highlighted content region.
- page.help: Dynamic help text, mostly for admin pages.
- page.content: The main content of the current page.
- page.sidebar_first: Items for the first sidebar.
- page.sidebar_second: Items for the second sidebar.
- page.footer: Items for the footer region.
- page.breadcrumb: Items for the breadcrumb region.

Now, main menu, secondary menu and breadcrumb have their own regions. If your theme does not declare any regions Drupal will assume a set of default regions. These regions correspond with what the default `core/modules/system/templates/page.html.twig` file expects, as well as two additional regions, page_top, and page_bottom, which are output in the default `html.html.twig` template.

If you declare any regions in your theme, even just one, all the default regions will no longer be applied and you assume responsibility for declaring any and all regions you want to use. Note: in most cases you'll want to make sure you declare the page_top and page_bottom regions in your theme since those are output by the `html.html.twig` file and some modules might expect them to be present.

### Review exercise

Add the copyright region to your bitingbugs subtheme. Enable the theme and see if the regions show up on the block layout page. What do you see?

## 10.4 TWIG templates

Twig is a PHPbased compiled templating language. When your web page renders, the Twig engine takes the template and converts it into a 'compiled' PHP template which it stores in a protected directory in `sites/default/files/php_storage/...` The compilation is done once. Template files are cached for reuse and are recompiled on clearing the Twig cache.

### 10.4.1 Working with Twig templates

Drupal allows you to override all of the templates that are used to produce HTML markup so that you can fully control the markup that is being output within a custom theme. There are templates for each page element ranging from the high level HTML to small fields.

### 10.4.2 Overriding templates

You can override Drupal core templates by adding templates to your theme folder that follow a specific naming convention. To override templates you:

- Locate the template you wish to override.
- Copy the template file from its base location into your theme folder.
- (optionally) Rename the template according to the naming conventions in order to target a more specific subset of areas where the template is used.
- Modify the template to your liking.

Once you copy a template file into your theme and clear the cache Drupal will start using your instance of the template file instead of the base version. You can find out what templates are in use for any part of the page by using the Twig debugging tools.

#### Rebuild cache

When working with theme hook suggestions, there is a possibility that Drupal uses its cache rather than the new templates as suggested. Clear the cache if you experience this problem. To clear the cache, you can do this through the interface, or with drush. The drush command has been changed and is now *drush cache-rebuild* or *drush cr* for short.

#### Overview of the template naming conventions

Say we want to add a template override for a page, drupal will look at the path and for example see `http://mysite.com/node/1`. Drupal will split this path up into components. In this case the components are **node** and **1**. And because we want to change a template for an entire page, the prefix will be

*page*. When the prefix is chosen drupal goes through the components to decide possible template names. For example if we want to change the *page* template for the `http://mysite.com/node/1` url we can change the following templates:

1. page.html.twig (This will change the template for all pages)
2. page–node.html.twig (This changes the template for all /node pages)
3. page–node–%.html.twig (This will change the template for all /node pages as well)
4. page–node–1.html.twig (This will change the template for the node with id 1)
5. page–front.html.twig (This will only change node/1 if it is the front page)

When the page is actually rendered, the last possibility is checked. If it exists, that possibility is used. Otherwise the next possibility up is checked, and so on. Of course, if none of the overriding possibilities exist, page.html.twig is the final possibility. This also explains why page–front.html.twig, if it exists, overrides any other possibilities for the front page: it is always the last possibility for the page designated as the front page. Besides page templates we can add a number of other templates to override more specific html, heres a list with examples of template names:

**HTML** The HTML template provides markup for the basic structure of the HTMLpage including the <head>, <title>and <body>tags. Base template: html.html.twig (base location: `core/modules/system/templates/html.html.twig`) Here is an example of how you may override the base template: html–node–id.html.twig

**Page** Pattern: page–[front—internal/path].html.twig Base template: page.html.twig (base location: `core/modules/system/templates/page.html.twig`) `http://mysite.com/node/1/edit` would result in the following suggestions:

1. page–node–edit.html.twig
2. page–node–1.html.twig
3. page–node.html.twig
4. page.html.twig

**Regions** Pattern: region–[region].html.twig Base template: region.html.twig (base location: `core/modules/system/templates/region.html.twig`)

**Blocks** Pattern: block–[module[–delta]].html.twig Base template: block.html.twig (base location: `core/modules/block/templates/block.html.twig`) *module* being the name of the module and *delta*, the internal id assigned to the block by the module. If you have a block created by views module with a view name *front_news* and display id *block_1* then the template would be:

block–views-block–front-news-block-1.html.twig (notice, when you have underscores in a display id or in a view name you have to transform them into a single dash) Be aware that module names are case sensitive in this context. For instance if your module is called *MyModule*, the most general template for this module would be *block–MyModule.html.twig*.

**Nodes** Pattern: node–[type—nodeid].html.twig Base template: node.html.twig (base location: `core/modules/node/templates/node.html.twig`) Template names are made based on these factors, listed from the most specific template to the least. Drupal will use the most specific template it finds:

1. node–nodeid.html.twig
2. node–type.html.twig
3. node.html.twig

Note that underscores in a content type's machine name are replaced by hyphens.

**Taxonomy terms** Pattern: taxonomy-term–[vocabulary-machine-name—tid].html.twig Base template: taxonomy-term.html.twig (base location: `core/modules/taxonomy/templates/taxonomyterm.html.twig`) Template names are made based on these factors, listed from the most specific template to the least. Drupal will use the most specific template it finds:

1. taxonomy-term–tid.html.twig
2. taxonomy-term–vocabulary-machine-name.html.twig
3. taxonomy-term.html.twig

Note that underscores in a vocabulary's machine name are replaced by hyphens.

**Fields** Pattern: field–[type—name[–content-type]—content-type].html.twig Base template: field.html.twig (base location: `core/modules/system/templates/field.html.twig`) Template names are made based on these factors, listed from the most specific template to the least. Drupal will use the most specific template it finds:

1. field–field-name–content-type.html.twig
2. field–content-type.html.twig
3. field–field-name.html.twig
4. field–field-type.html.twig

Note that underscores in a Field's machine name are replaced by hyphens. Also remember to include *field* in custom field names, e.g: field–field-phone.html.twig

**Comments** Pattern: comment–node-[type].html.twig Base template: comment.html.twig (base location: `core/modules/comment/template/comment.html.twig`

**Forums** Pattern: forums–[[container—topic]–forumID].html.twig Base template: forums.html.twig (base location: `core/modules/forum/templates/forums.html.twig`).

**Maintenance page** Pattern: maintenance-page–[offline].html.twig Base template maintenance-page.html.twig (base location: `core/modules/system/templates/maintenancepage.html.twig`).

**Search result** Pattern: search-result–[searchType].html.twig Base template: search-result.html.twig (base location: `core/modules/search/templates/searchresult.html.twig`).

### 10.4.3 The template file contents

Have a look at the `page.html.twig` template. You can find it at `core/themes/classy/templates/layout/`. In figure 10.14 you can see the header.

```
62  <header role="banner">
63    {% if logo %}
64      <a href="{{ front_page }}" title="{{ 'Home'|t }}" rel="home">
65        <img src="{{ logo }}" alt="{{ 'Home'|t }}"/>
66      </a>
67    {% endif %}
68
69    {% if site_name or site_slogan %}
70      <div class="name-and-slogan">
71
72        {# Use h1 when the content title is empty #}
73        {% if title %}
74          <strong class="site-name">
75            <a href="{{ front_page }}" title="{{ 'Home'|t }}" rel="home">{{ site_name }}</a>
76          </strong>
77        {% else %}
78          <h1 class="site-name">
79            <a href="{{ front_page }}" title="{{ 'Home'|t }}" rel="home">{{ site_name }}</a>
80          </h1>
81        {% endif %}
82
83        {% if site_slogan %}
84          <div class="site-slogan">{{ site_slogan }}</div>
85        {% endif %}
86      </div>{# ./name-and-slogan #}
87    {% endif %}
88
89    {{ page.header }}
90  </header>
```

Figure 10.14: page.html.twig content.

As you can see a template file is made up of standard html and some strange tags. The Twig engine is responsible for interpreting these tags. There are three types of tags:

- {{ *These* }} are for printing content, either explicitly or via functions.
- {% *These* %} are for executing statements.
- {# *These* #} are for comments.

### 10.4.4 Printing variables and regions

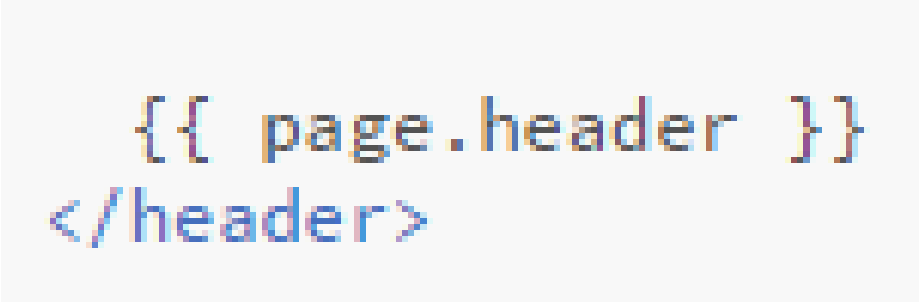



Figure 10.15: Printing variables and regions.

### 10.4.5 Executing statements

```
{% if title %}
  <strong class="site-name">
    <a href="{{ front_page }}" title="{{ 'Home'|t }}" rel="home">{{ site_nar
  </strong>
{% else %}
  <h1 class="site-name">
    <a href="{{ front_page }}" title="{{ 'Home'|t }}" rel="home">{{ site_nar
  </h1>
{% endif %}

{% if site_slogan %}
  <div class="site-slogan">{{ site_slogan }}</div>
{% endif %}
```

Figure 10.16: Executing statements.

### 10.4.6 Comments


Figure 10.17: Comments

### 10.4.7 Overriding a template

In this section we are going to override the page template in our bitingbugs theme. Copy the template file `page.html.twig` from `/core/themes/classy/templates/layout/` to `/themes/bitingbugs/templates/layout/`. Open the template file. You can see that the different regions that we defined in our

.info.yml file are printed in certain locations on this page. If you remember, our **copyright** region did not show up on the regions page. To make it show up we will have to add it to our page template. Add The following code above the closing main tag:

```
  {% if page.copyright %}
  <aside class="layout-copyright" role="complementary">
      {{ page.copyright }}
  </aside>
  {% endif %}

</main>
```

Figure 10.18: Copyright region

Clear the drupal caches and to the block region demonstration on your site. There you will see the **Copyright** region show up between the **Sidebar first** and **Footer** regions.

Next we will wrap the footer content into a div container so we can give it af full width layout later:

```
142     <div class="foot-background">
143         <div class="layout-container">
144             {% if page.footer %}
145                 <footer role="contentinfo">
146                     {{ page.footer }}
147                 </footer>
148             {% endif %}
149         </div>
150     </div>
151
```

Figure 10.19: Page footer wrapper.

### 10.4.8 Twig filters

Filters in Twig can be used to modify variables. Filters are separated from the variable by a pipe symbol (—) and may have optional arguments in parentheses. Multiple filters can be chained. The output of one filter is applied to the next. Example: ponies—safe_join(", ")—lower The list of filters that can be used in Twig templates for Drupal consists of all the filters in the Twig engine as well as some Drupal specific filters.

#### Translation filters

The t filter will run the variable through the Drupal t() function, which will return a translated string. This filter should be used for any interface strings manually placed in the template that will appear for users.

```
{% if logo %}
  <a href="{{ front_page }}" title="{{ 'Home'|t }}" rel="home">
    <img src="{{ logo }}" alt="{{ 'Home'|t }}"/>
  </a>
{% endif %}
```

Figure 10.20: Translation filter.

This means that if we add this filter, we should be able to translate this string in the translation interface.

#### passthrough, and placeholder

In order to safely escape all of the Twig variables detected in an % trans % tag, the variables are sent with an @ prefix by default. To passthrough a variable (!) or use as a placeholder (%), the passthrough or placeholder filters are used. Passthrough gets no sanitization or formatting and should only be used for text that has already been prepared for HTML display. placeholder gets escaped to HTML and formatted using drupal_placeholder(), which makes it display as emphasized text.

## 10.5 Debugging Twig templates

The Twig engine provides options for configuring debugging, automatic reloading (recompiling) of templates, and caching compiled templates in the filesystem. By default, the Twig theming engine compiles templates into PHP code and stores the compiled code in memory. Compiled code is unsuitable for development, since changes in Twig templates are not immediately updated in your Drupal site. Twig cache can be cleared through Drupal's clear cache interface, but for ongoing development it's easier to change Drupal's settings so that Twig doesn't cache anything at all. The Drupal 8 implementation also adds an additional tool that allows you to locate the template that outputs the markup.

### 10.5.1 Enable debugging

To enable debugging locate the `/sites/default/default.services.yml` file. Copy this file and rename it to services.yml. Edit the services.yml file and change the following parameters:

**debug** true (Turn on debugging)

**auto_reload** true (Automatically recompile Twig templates when the source changes)

**cache** false (Disable caching)

Now clear the drupal cache. Go to **Configuration** → **Development** → **Performance** → **Clear Cache** or use the **drush cache-rebuild** command.

## 10.6 Review exercises

In this exercise you are going to create a new subtheme for our bitingbugs example. Our new theme will be based on the Bartik theme.

1. Now create a new folder for your theme and put the correct .info.yml file inside. You can use the default Bartik regions.

2. enable your new subtheme.

3. Move the **Powered by drupal**, **User account menu**, **Footer menu**, **Help**, **Tools** and **User login** blocks to the **None** region.

4. We would like to change the size of the recipe titles on the homepage. Use your browser's development tools to locate the twig template where the title is generated. Override the template and wrap the title inside H2 tags.

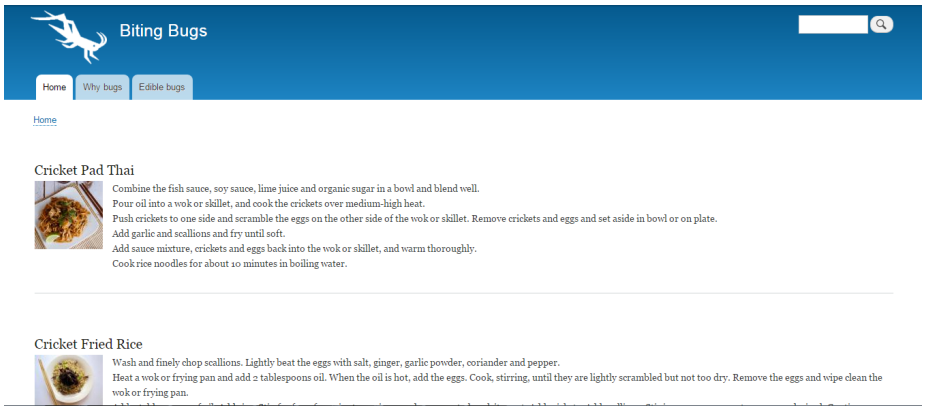
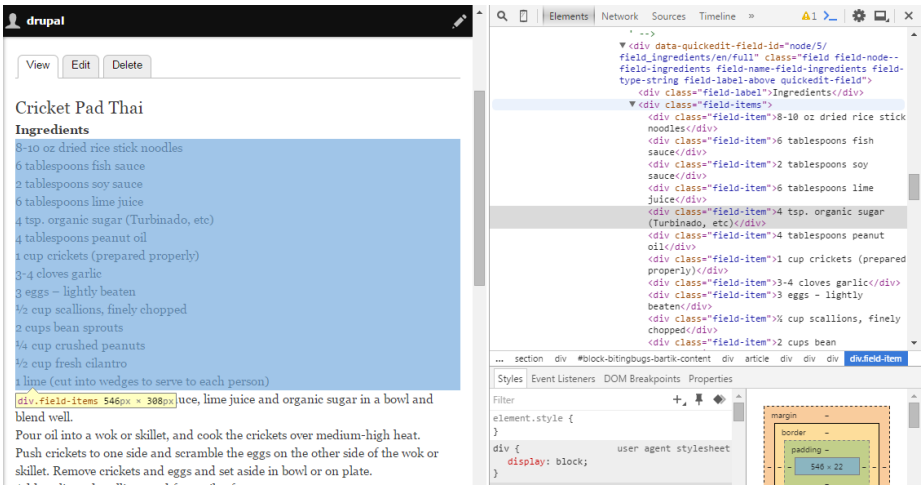



Figure 10.21: We changed the title by wrapping it in an H2 tag.

5. Next we are going to change the recipe page. Click one of the recipe titles on the homepage. Inspect the html behind the ingredients list.




As you can see right now the ingredients list is a DIV element with another DIV for each recipe. Override the right template to change it to an html list. The resulting HTML should look like this:

```
▼<div data-quickedit-field-id="node/5/field_ingredients/en/full" class="field field-node--
field-ingredients field-name-field-ingredients field-type-string field-label-above
quickedit-field">
    <div class="field-label">Ingredients</div>
  ▼<ul class="field-items">
      <li class="field-item">8-10 oz dried rice stick noodles</li>
      <li class="field-item">6 tablespoons fish sauce</li>
      <li class="field-item">2 tablespoons soy sauce</li>
      <li class="field-item">6 tablespoons lime juice</li>
      <li class="field-item">4 tsp. organic sugar (Turbinado, etc)</li>
      <li class="field-item">4 tablespoons peanut oil</li>
      <li class="field-item">1 cup crickets (prepared properly)</li>
      <li class="field-item">3-4 cloves garlic</li>
      <li class="field-item">3 eggs - lightly beaten</li>
      <li class="field-item">½ cup scallions, finely chopped</li>
      <li class="field-item">2 cups bean sprouts</li>
      <li class="field-item">¼ cup crushed peanuts</li>
      <li class="field-item">½ cup fresh cilantro</li>
      <li class="field-item">1 lime (cut into wedges to serve to each person)</li>
    </ul>
  </div>
```

6. HINT: make sure you always use the most specific filename for your overridden template

7. HINT: Use the debug comments to locate the correct template file to override.

## 10.7 CSS and JavaScript

In Drupal 8 JavaScript and CSS files are added through libraries.

### 10.7.1 Defining a library

Add a `[MyTheme].libraries.yml` file to your theme folder. The libraries file should have the following structure:

```
1  global-styling:
2    css:
3      theme:
4        css/mystyles.css: {}
5    js:
6      js/myscript.js: {}
```

This example assumes that the actual JavaScript myscript.js is located in the subfolder js of your theme. However, remember that Drupal 8 no longer loads jQuery on all pages by default Drupal 8 only loads what's necessary. Therefore, we must declare that our theme's myscript library has a dependency on the library that contains jQuery. It is neither a module nor a theme that provides jQuery, it's Drupal core: core/jqueryis the dependency we want to declare. (This is an extension name followed by a slash, followed by the library name, so if some other library wanted to depend on our cuddly-sliderlibrary, it'd have to declare a dependency on Mytheme/cuddlyslider, because Mytheme is the name of of our theme.) So, to ensure jQuery is available for js/myscript.js, we update the above to:

```
1  global-styling:
2    css:
3      theme:
4        css/mystyles.css: {}
5    js:
6      js/myscript.js: {}
7    dependencies:
8      - core/jquery
```

You can specify the media type for each css file, for example if you have a css file for the print layout then add the following line:

```yaml
global-styling:
  css:
    theme:
      css/mystyles.css: {}
      css/print.css: { media: print }
  js:
    js/myscript.js: {}
  dependencies:
    - core/jquery
```

### 10.7.2 Attaching libraries to page(s)

#### Attaching to all pages

To attach a library to all pages that use your theme, declare it in your theme `*.info.yml` file, under the libraries key:

```yaml
name: bitingbugs_bartik
type: theme
base theme: bartik
description: 'A sub theme for our bitingbugs site based on
bootstrap business'
package: Custom
core: 8.x
libraries:
  - bitingbugs/global-styling
stylesheets-remove:
  -core/assets/vendor/normalize-css/normalize.css
```

You can list as many libraries as you want, all of them will be loaded on every page. Then clear the cache so that the new information is loaded into Drupal.

#### Attaching a library to a subset of pages

In some cases, you do not need your library to be active for all pages, but just a subset of pages. For example, you might need your library to be active only when a certain block is being shown, or when a certain node type is being displayed.

First we define our library `[themename].libraries.yml`:

```yaml
global-styling:
  css:
    theme:
      css/mystyles.css: {}
      css/print.css: { media: print }
  js:
    js/myscript.js: {}
  dependencies:
    - core/jquery
```

A theme can make this happen by implementing a THEME_preprocess_HOOK() function in the .theme file, replacing *THEME* with the machine name of your theme and *HOOK* by the machine name of the theme hook. For instance, if you want to attach JavaScript to the maintenance page, the *HOOK* part is "maintenance_page", and your function would look like this:

```php
<?php

function
bitingbugs_bartik_preprocess_maintenance_page(&$variables){
  $variables['#attached]['library'][] =
'bitingbugs_bartik/custom-library';
}
```

You can do something similar for other theme hooks, and of course your function can have logic in it  for instance to detect which block is being preprocessed in the *block* hook, which node type for the *node* hook, etc.

### Attaching a library in a twig template

You can also attach a library in a twig template by using the attach_library() twig function So in any `*.html.twig`:

```
{% if threaded %}
  {{ attach_library('classy/drupal.comment.threaded') }}
{% endif %}
```

### Inline JavaScript

Inline JavaScript is discouraged. It's recommended to put the JS you want to use inline in a file instead, because that allows that JavaScript to be cached on the client side. It also allows JavaScript code to be reviewed and linted.

## 10.8 Review exercises

In the next exercises we are going to change the look of our site.

1. Create a css folder in bitingbugs_bartik theme folder. In this folder create a mytheme.css file.

2. Add the `mytheme.css` file to the `.libraries` file.

3. We are going to create a new view which displays a list of all recipes. Go to **Structure** → **Views** → **Create View**. Add a page view which displays all recipes sorted by newest first. Add a link to your page in the main navigation.

4. Change the image style to **None, (Original Image)**. Do this in the teaser view of our recipe content type.

5. The next part is pretty difficult. We are going to change the structure of our recipe html page by overriding different templates. The following image shows the structure we are trying to create:


   First make sure you have enabled debugging in your `services.yml` file. Look for the div with the class *view-content* containing four article elements. Use the debug information to override the right templates until you get the correct structure.

6. Next we are going to change the `mytheme.css` file to update our layout. Change the css file to make your page look as follows:

Recipes


### 10.8.1 JavaScript

Now that weve seen how to add js in different ways, its time to look at some best practices.

#### JavaScript closures

It's best practice to wrap your code in a closure. A closure is nothing more than a function that helps limit the scope of variables so you don't accidentally overwrite global variables.

```
1  //Define a new function.
2  (function(){
3    //Variables defined here will not affect the global scope.
4    var window = "Whoops, at leat I only broke my code!";
5    console.log(window);
6  //The extra set of parenthesis here says run the function we just created.
7  }());
8  //Our wacky code inside our closure doesn't affect everyone else.
9  console.log(window);
```

A closure can have one other benefit, if we pass jQuery in as a parameter we can map it to the $ shortcut allowing us to use $() without worrying if jQuery.noConflict() has been called.

```
1  // We define a function that takes one parameter $
2  (function ($){
3    //Use jQuery with the shortcut:
4    console.log($.support);
5  //Here we immediately coll the function with jQuery as the parameter.
6  }(jQuery));
7
```

### Behaviors

If youre adding JavaScript to your custom module it is very easy and tempting to simply add it like this:

```
jQuery(document).ready(function($){
    alert("I do work!");
});

```

Now this code works perfectly fine but what if your JavaScript needs to be executed on page load and after an AJAX request? Imagine you have a view that uses Views Infinite Scroll and you want add a CSS class to every result like this:

```
jQuery(document).ready(function($){
    $('.view-display-id-page .views-row').addClass('new-class');
});


```

This will work for the results that are displayed initially but for all the results that are loaded by Infinite Scroll's AJAX call the class is not added. Thats where Drupal behaviors come in handy. The behaviors will be executed on every request including AJAX requests, so let's do the equivalent of the code above but this time using this method:

```
//Using the closure to map jQuery to $.
(function($){
    Drupal.behaviors.infinitScroll.AddClass = {
        attach: function(context, settings){
            $('.view-display-id-page .views-row').addClass('new-class');
        }
    };
}(jQuery));

jQuery(document).ready(function($){
    $('.view-display-id-page .views-row').addClass('new-class');
});

```

I admit that was quick  so here are some explanations:

- infiniteScrollAddClass: This is your namespace and should be unique. For example, this is typically the name of your module, but it isn't mandatory.
- context: This is actually really cool, on page load the context will contain the entire document and after an AJAX request will have all the newly

loaded elements. This way you can treat content that is loaded in via AJAX differently than others.

- settings: This contains information passed on to JavaScript via PHP, it is similar to accessing it via Drupal.settings. There obviously are cases where some functionality should not be executed on every request. In such a case its great to use jQuery's .once() method. So let's say we want to give all the initially loaded results in our view an additional class, for something like this we would proceed like so:

```
3    Drupal.behaviors.infinitScroll.AddClass = {
4        attach: function(context, settings){
5            //these are the elements loaded in first
6            $('.view-display-id-page').once(function(){
7                $(this).find('.views-row').addClass('i-was-here-first');
8            });
9        }
10   };
11
12   //everybody
13   $('.view-display-id-page .views-row').addClass('new-class');
```

This will add the class *iwasherefirst* to all the view results present on page load, everybody else joining in via AJAX will just get the *newclass* class.