



Learn by doing: less theory, more results

# Drupal 8 Development

*Second Edition*

Develop your programming skills by creating engaging websites using Drupal 8

## *Beginner's Guide*

Neeraj Kumar  
Edward Crompton  
Malabya Tewari

Tassos Koutlas  
Krishna Kanth  
Kurt Madel

Samuel Keen  
Rakesh James

[PACKT] open source<sup>®</sup>  
community experience distilled  
PUBLISHING

# **Drupal 8 Development**

## **Beginner's Guide**

### ***Second Edition***

Develop your programming skills by creating engaging websites using Drupal 8

**Neeraj Kumar**  
**Tassos Koutlas**  
**Samuel Keen**  
**Edward Crompton**  
**Krishna Kanth**  
**Rakesh James**  
**Malabya Tewari**  
**Kurt Madel**



**open source**   
community experience distilled

BIRMINGHAM - MUMBAI

# **Drupal 8 Development Beginner's Guide**

## ***Second Edition***

Copyright © 2016 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: May 2012

Second edition: June 2016

Production reference: 1280616

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78528-488-5

[www.packtpub.com](http://www.packtpub.com)

# Credits

**Authors**

Neeraj Kumar  
Tassos Koutlas  
Samuel Keen  
Edward Crompton  
Krishna Kanth  
Rakesh James  
Malabaya Tewari  
Kurt Madel

**Reviewer**

Partha Bose

**Commissioning Editor**

Amarabha Banerjee

**Acquisition Editor**

Tushar Gupta

**Content Development Editor**

Arun Nadar

**Technical Editor**

Vivek Arora

**Copy Editor**

Vikrant Phadke

**Project Coordinator**

Ritika Manoj

**Proofreader**

Safis Editing

**Indexer**

Hemangini Bari

**Graphics**

Jason Monteiro

**Production Coordinator**

Melwyn Dsa

**Cover Work**

Melwyn Dsa

# About the Authors

**Neeraj Kumar** is a Drupal architect, author, project manager, and overall geeky guy who enjoys using technology for bringing value to businesses. He is also the chief engagement officer of Valuebound, an India-based Drupal consulting firm that provides enterprise Drupal solutions for media and product companies.

He earned his bachelor's degree in architecture from the Indian Institute of Roorkee. Yes, you heard it right; he is an architect by degree training and used to design buildings. But nowadays, he is more into architecting Drupal solutions for enterprises and advocating best practices, along with contributing to the awesome Drupal community. You can follow Neeraj on Twitter at <http://twitter.com/neerajskydiver> or his LinkedIn profile at <https://in.linkedin.com/in/neerajskydiver>.

---

I would like to thank the entire team at Valuebound who made this book possible. I would also like to thank my wife, Puja, and our little son, Aaryan, for allowing me time to work on this book. Thank you for being cheerful and happy even when I spent late evenings working.

---

**Tassos Koutlas** is a senior technical consultant at Cameron & Wilding in London. He has over 13 years of experience in producing web-based projects and machine learning / image processing algorithms, and administering IT systems. He has worked in commercial and research-based environments in the UK, Greece, and Italy. Tassos has successfully delivered projects such as the National Baseball Hall of Fame redevelopment, where he helped develop a better way to deliver content online, utilizing the Hall's vast asset collection. He has been part of award-winning projects such as the Qatar Museums website, where he helped extend their website with an events calendar and a blog section, and has been supporting MoMA online collections. More recently, he has been helping Investors in People embark on becoming an agile organization.

Prior to joining Cameron and Wilding, Tassos had been a part of the team working at the Southbank Centre, providing support at multiple levels in their digital transformation efforts. His previous projects include Q base R&D, a start-up focused on personalized skin cancer prevention, and the Athens 2004 Olympic games. Tassos graduated from the University of Manchester with a BSc Hons in computing science and holds a PhD in image processing and machine learning. He has been working with Drupal for 10 years. He is an Acquia certified developer and certified Scrum Master.

---

I would like to thank my wife, Fani, for her constant support of my endeavors.

---

**Samuel Keen** first found an interest in web development while traveling abroad, when he learned how to build an online photography portfolio. A trained musician, he found that the same interests drove his web development: "Both music and the Web have frameworks that give us almost endless abilities to build and be creative." Returning to the UK, he began designing and developing websites for people in creative industries. Since 2011, Samuel has worked almost exclusively with Drupal. He is particularly interested in the frontend, UX, and automated workflows, with a passion for cutting edge Sass.

He currently works as a frontend developer for Cameron & Wilding, a Drupal development agency based in London. Samuel has worked on projects for The Economist, the Imperial War Museums, London South Bank University, and The Telegraph.

---

My endless love and thanks to my wife, Romilly.

---

**Edward Crompton** has been developing with Drupal since he was first hooked by version 5 in 2007. His adventures in open source software development have taken him from the so-called Silicon Roundabout in London to the villages of South India, where he built an application for gathering data about school building projects. He's now based back in London, where he can often be found riding a rusty bicycle and digging in his rented potato patch.

Edward works as a Drupal developer for Cameron & Wilding Ltd. He's a keen supporter of Drupal community events in London and likes to exchange knowledge about Drupal at meetups and in the pub.

---

I'd like to thank Gorka Guridi and Lucia Otoyo for their valuable feedback while writing the chapter and for shaming me into doing something about my dirty coffee cups.

---

**Krishna Kanth** is a Drupal developer and an active contributor to the Drupal community who has been building websites since he was a student at engineering college. He is a specialist in the Drupal backend and equally loves to work with other PHP frameworks.

He is a graduate in information technology from Jawaharlal Nehru Technological University, Anantapur, Andhra Pradesh, India. Krishna started his career as a freelancer and worked with many NGOs. Now he works as a senior Drupal consultant and developer at Valuebound, India. You can follow him on Twitter at [https://twitter.com/\\_krishna\\_kanth](https://twitter.com/_krishna_kanth) or his LinkedIn profile at <https://in.linkedin.com/in/krishnakanthp>.

**Rakesh James** is a Drupal developer, evangelist, enthusiast, and contributor to the Drupal community. He is a specialist in Drupal who simply loves to work with Drupal again and again. He is one of those who believes and thanks God for Drupal on a daily basis, because he believes that if Dries hadn't created Drupal, he may not have achieved his current lifestyle. So he is always looking to evangelize, train, and mentor newbies to Drupal.

Rakesh graduated from the Government Engineering College, Thrissur, which is one of the finest engineering institutes in Kerala. He started his career as a software programmer in PHP and Drupal. Currently, he is hand in hand with Valuebound as a senior Drupal consultant, developer, contributor, mentor, trainer, and architect for Drupal projects.

**Malabya Tewari** is a full-stack Drupal developer, Drupal evangelist, trainer, and open source enthusiast who likes to reinvent and improve his working environment continuously, mainly because of his unsatisfied soul which seeks to achieve perfection. He is an active contributor to the Drupal community via code, as well as by organizing Drupal meetups and "Drupal in a day" workshops in Bangalore.

A graduate in computer science from Sikkim Manipal Institute of Technology, who started his career as an intern in a Drupal firm, Malabya now works as a Drupal consultant and developer at Valuebound, which is a Drupal shop that delivers enterprise Drupal solutions for media and product companies.

**Kurt Madel** is a senior manager and developer for Captech Consulting in Richmond, Virginia, USA. He has worked on open source CMS projects for over 6 years. Kurt contributes regularly to Drupal.org and is the maintainer of several modules. In addition to Drupal, he has been doing Java EE development since 2000 and has focused on mobile web development over the last 2 years. When he is not writing or programming, Kurt enjoys cycling and spending time with his wife and four boys.

# About the Reviewer

**Partha Bose** is an MCA, sun certified. He has been working in web development for the last 10 years.

He has been working with Drupal since late 2011. He soon became as interested in the challenge of fixing bugs and adding features to Drupal core and contributed modules. He has been using Drupal as the primary platform for creating beautiful and feature-rich sites. Partha's passion for Drupal is evident in his obsession with evangelizing the platform and his enthusiasm when speaking with clients about the possibilities of what they can accomplish by using Drupal.

He is currently working for PWC India as a senior Drupal developer. He played a crucial role of being a Drupal SME for the successful development of web-based applications.

---

I'd like to thank to my wife, Mrs. Modhumita Bose, for her active support in producing this book.

---

# **www.PacktPub.com**

## **eBooks, discount offers, and more**

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [customercare@packtpub.com](mailto:customercare@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

## **Why subscribe?**

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser



# Table of Contents

|   |            |
|---|------------|
| <b>Preface</b>  | <b>vii</b> |
| <b>Chapter 1: Setting Up a Drupal Development Environment</b> | <b>1</b>   |
| Installing Drupal for local development                       | 1          |
| Time for action – installing Drupal using Acquia Dev Desktop  | 2          |
| Installing Drupal the localhost way                           | 7          |
| Time for action – installing a Mac OS X AMP stack             | 7          |
| Time for action – installing a Windows AMP Stack              | 9          |
| PHP configuration   | 11         |
| Time for action – modifying the php.ini settings              | 11         |
| Modifying the MySQL my.cnf settings                           | 12         |
| Time for action – setting up MySQL for Drupal                 | 12         |
| Time for action – creating an empty MySQL database            | 13         |
| Installing Git  | 13         |
| Time for action – installing Git for Mac OS X                 | 14         |
| Time for action – installing Git for Windows                  | 14         |
| Installing Drush  | 15         |
| Time for action – installing Drush for Mac OS X               | 16         |
| Time for action – installing Drush for Windows                | 16         |
| Installing Drupal 8   | 16         |
| Time for action – installing Drupal 8                         | 17         |
| Installing the PHPStorm IDE                                   | 21         |
| Time for action – installing the PHPStorm IDE                 | 21         |
| Time for action – creating a new PHPStorm project             | 22         |
| Drupalize PHPStorm IDE  | 23         |
| Drupal from a developer's perspective                         | 24         |
| Installing Vagrant  | 25         |
| Time for action – installing Vagrant                          | 25         |
| Summary   | 29         |

*Table of Contents*

---

|  |            |
|--|------------|
| <b>Chapter 2: Custom Module Development</b>  | <b>31</b>  |
| Creating custom Recipe content type  | 32         |
| Time for action – creating custom content type   | 32         |
| Time for action – adding a new recipe  | 39         |
| OOP concepts in Drupal   | 42         |
| Time for action – developing a custom module in Drupal 8   | 46         |
| Time for action – developing custom field formatter  | 51         |
| Test-driven development (TDD)  | 58         |
| PHPUnit tests for Drupal classes   | 60         |
| Functional tests   | 60         |
| Time for action – writing and testing functional test from our d8dev<br>custom module            | 60         |
| Summary  | 63         |
| <b>Chapter 3: Drupal Views and Configuration Management</b>                                      | <b>65</b>  |
| A quick introduction to Views  | 65         |
| Time for action – creating a recipe listing block using views                                    | 66         |
| Configuration management in Drupal 8   | 72         |
| Using the Configuration Management interface   | 73         |
| Time for action – importing, exporting, and synchronizing configurations                         | 74         |
| Working of Configuration Management in Drupal 8  | 83         |
| Changing the active configuration storage  | 84         |
| Introducing the Devel module   | 86         |
| Installing the Devel module  | 86         |
| Time for action – generating dummy content using the devel_generate module                       | 87         |
| Summary  | 88         |
| <b>Chapter 4: Introduction to the Field Types API and Developing<br/>the Custom Field Module</b> | <b>89</b>  |
| Introducing the NutritionInformation module  | 89         |
| Time for action – developing a custom module for a compound<br>NutritionInformation field        | 90         |
| Time for action – updating the Recipe content type to use the<br>NutritionInformation field      | 106        |
| Summary  | 107        |
| <b>Chapter 5: Theming in Drupal 8</b>  | <b>109</b> |
| What is a theme?   | 109        |
| Time for action – creating a sub-theme   | 111        |
| An overview of Bartik  | 113        |
| Mobile first, responsive themes  | 114        |

---

*Table of Contents*

|  |            |
|--|------------|
| <b>Time for action – installing Drush</b>                                  | <b>116</b> |
| <b>Time for action – Adding assets to your theme</b>                       | <b>117</b> |
| <b>Time for action – calling assets on specific pages</b>                  | <b>121</b> |
| <b>Introduction to templating and Twig</b>                                 | <b>126</b> |
| Theme hook suggestions   | 128        |
| File and function names  | 129        |
| Brackets syntax  | 130        |
| Rendering  | 130        |
| Filters  | 131        |
| Control structures   | 131        |
| <b>Debugging Twig</b>  | <b>132</b> |
| HTML comments in markup  | 132        |
| Debugging variables  | 132        |
| Kint   | 133        |
| <b>Time for action – Twig in practice</b>                                  | <b>133</b> |
| <b>Time for action – understanding the benefits of contributed modules</b> | <b>139</b> |
| <b>What are contributed modules?</b>                                       | <b>143</b> |
| How do I know whether a module is safe to use?                             | 143        |
| Is it better to use a contrib module or custom code?                       | 144        |
| <b>Summary</b>   | <b>144</b> |
| <b>Chapter 6: Enhancing the Content Author's User Experience</b>           | <b>145</b> |
| A quick introduction to CKEditor in Drupal 8                               | 145        |
| Configuring CKEditor profiles  | 146        |
| Time for action - adding some buttons to the basic HTML profile            | 147        |
| Time for action - exporting CKEditor configuration                         | 151        |
| Adding a new CKEditor profile  | 153        |
| Time for action - creating a text-only control profile for anonymous users | 153        |
| Classic editor and inline editing  | 156        |
| Time for action – using inline editing                                     | 156        |
| Adding widgets to CKEditor   | 157        |
| Introduction to the Block API for Drupal 8                                 | 160        |
| Time for action – creating a block to aid the authoring experience         | 161        |
| Time for action – including default configuration in your module           | 169        |
| Summary  | 170        |
| <b>Chapter 7: Adding Media to Our Site</b>                                 | <b>171</b> |
| Introduction to the File entity module                                     | 172        |
| Working with dev versions of modules                                       | 172        |
| Time for action – installing a dev version of the File entity module       | 172        |
| A new recipe for our site  | 173        |

---

*Table of Contents*

---

|  |            |
|--|------------|
| Time for action – adding a Recipe images field to our Recipe content type                              | 175        |
| Creating a custom image style  | 180        |
| Time for action – adding a custom image style through the image style administrative page              | 180        |
| Time for action – creating a programmatic custom image style   | 182        |
| Integrating the Colorbox and File entity modules   | 184        |
| Time for action – installing the Colorbox module   | 184        |
| Working with Drupal issue queues   | 187        |
| Time for action – creating an issue for the Colorbox module  | 188        |
| Summary  | 190        |
| <b>Chapter 8: How Does It Taste? – Getting Feedback</b>  | <b>191</b> |
| Introduction to the Drupal contact form  | 191        |
| Time for action – enabling and configuring the core contact form                                       | 191        |
| Adding placeholder text to our contact form  | 193        |
| Using configurations to add placeholder text to the contact form                                       | 193        |
| Time for action – adding placeholder text to our site contact form                                     | 193        |
| Using custom code to add placeholder text to the Name and Email fields                                 | 196        |
| Time for action – adding placeholder text to Name and Email fields                                     | 196        |
| Time for another recipe!   | 198        |
| Colorbox file enhancements   | 200        |
| Time for action – enhancing the Colorbox module with image title and alt captions                      | 201        |
| Contributing our code to Drupal  | 206        |
| Time for action – creating a patch and uploading it on the Drupal issues queue                         | 206        |
| Recipe reviews with comments   | 208        |
| Time for action – configuring comments as recipe reviews   | 208        |
| Time for action – enhancing the liking system using comments and views                                 | 214        |
| Summary  | 224        |
| <b>Chapter 9: Advanced Views Development</b>   | <b>225</b> |
| Views revisited – advanced configuration   | 226        |
| Random top rated recipe block  | 226        |
| Time for action – building a random top rated recipe block with Views                                  | 226        |
| Taxonomy-based View with tabs  | 232        |
| Time for action – creating a cuisine vocabulary to organize recipes                                    | 232        |
| Time for action – creating a Recipes by cuisine type Views block                                       | 235        |
| Time for action – installing and using the Views Field View module for our Recipe by Cuisine Type View | 239        |
| Tabbed Views display   | 244        |
| Time for action – developing a Views style plugin for semantic tabs                                    | 245        |
| Time for another Recipe  | 255        |

---

---

*Table of Contents*

|   |            |
|---|------------|
| <b>Contributing the Views semantic tabs module to Drupal</b>  | <b>257</b> |
| <b>Time for action – creating a sandbox for the views semantic tabs module</b>                                  | <b>258</b> |
| <b>Summary</b>  | <b>263</b> |
| <b>Chapter 10: Drupal Project Management and Collaboration</b>  | <b>265</b> |
| <b>Rotating banners with the Views Slideshow module</b>   | <b>265</b> |
| <b>Time for action – installing the Views Slideshow module</b>  | <b>266</b> |
| <b>Creating a rotating banner with Views Slideshow</b>  | <b>267</b> |
| <b>Time for action – creating a banner using the Views Slideshow module</b>                                     | <b>267</b> |
| <b>Time for action – creating a new image style for images in our rotating recipe banner</b>                    | <b>272</b> |
| <b>Enhancing the appearance of our front banner with a pager and CSS</b>  | <b>274</b> |
| <b>Time for action – updating the front banner view to include a slide show pager</b>                           | <b>274</b> |
| <b>Time for another recipe</b>  | <b>281</b> |
| Promoting a sandbox project to a full project   | 283        |
| <b>Time for action – creating README.txt and pushing to the sandbox</b>   | <b>283</b> |
| <b>Time for action – promoting the Views semantic module to a full project on Drupal.org</b>                    | <b>285</b> |
| <b>Introducing the Features module</b>  | <b>290</b> |
| <b>Time for action – installing the Features module</b>   | <b>291</b> |
| Recipe feature by the Features module   | 292        |
| <b>Time for action – using the Features module to export the Recipe content type and related configurations</b> | <b>292</b> |
| <b>When to use core Configuration Management compared to Features</b>   | <b>296</b> |
| <b>Summary</b>  | <b>297</b> |
| <b>Chapter 11: Searching Your Site with the Search API Module</b>   | <b>299</b> |
| <b>The Drupal core search</b>   | <b>299</b> |
| <b>The Search API module</b>  | <b>300</b> |
| <b>Time for action – basic installation and configuration of the Search API module</b>                          | <b>301</b> |
| An explanation of search servers and search indexes   | 303        |
| Search server   | 303        |
| Search index  | 305        |
| Fields  | 306        |
| Processors  | 307        |
| Populating your search index  | 309        |
| <b>Exposing the search to users</b>   | <b>311</b> |
| Altering the search display   | 312        |
| Excluding entities from being indexed   | 314        |
| <b>Installing Apache Solr as the search backend</b>   | <b>315</b> |
| Installing Solr 4.x on a virtual machine with Vagrant and Puppet  | 316        |

---

---

*Table of Contents*

---

|   |            |
|---|------------|
| <b>Time for action – creating and configuring your virtual machine</b>              | <b>317</b> |
| Installing Solr 5.x manually on Ubuntu 14.04  | 323        |
| <b>Time for action – installing and configuring Solr on Ubuntu</b>                  | <b>324</b> |
| <b>Securing Apache Solr with Uncomplicated Firewall</b>                             | <b>327</b> |
| <b>Time for action – configuring Uncomplicated Firewall</b>                         | <b>327</b> |
| <b>The Search API Solr module</b>   | <b>328</b> |
| <b>Time for action – configuring Drupal to use Apache Solr</b>                      | <b>328</b> |
| Using the read-only mode  | 332        |
| <b>Search facets</b>  | <b>332</b> |
| <b>Time for action – building faceted search blocks</b>                             | <b>333</b> |
| <b>Summary</b>  | <b>341</b> |
| <b>Chapter 12: RESTful Web Services in Drupal</b>                                   | <b>343</b> |
| <b>Introduction to web services</b>   | <b>343</b> |
| <b>Introduction to REST</b>   | <b>344</b> |
| <b>Headless Drupal</b>  | <b>346</b> |
| When to decouple Drupal or when to use Headless Drupal                              | 348        |
| <b>RESTful web services in Drupal</b>   | <b>349</b> |
| RESTful APIs in Drupal  | 349        |
| <b>Time for action – getting all the recipe types</b>                               | <b>350</b> |
| <b>Time for action – creating an API to get all the recipes under a recipe type</b> | <b>357</b> |
| <b>Time for action – consuming RESTful web services using AngularJS</b>             | <b>361</b> |
| Summary   | 363        |
| <b>Appendix: Pop Quiz Answers</b>   | <b>365</b> |
| <b>Index</b>  | <b>367</b> |

---

# Preface

The traditional Web, where websites are viewed by people on desktop computers, is obsolete. Web content is now viewed in a myriad of ways by users with mobile phones, tablets, and even watches. Content on the Internet may not be consumed directly by humans at all—web services now power apps on our phones and almost every device that's part of the emerging and much lauded Internet of Things.

Drupal is evolving to reflect the changes we're seeing on the Internet as a whole, and Drupal 8 represents a big technological leap forward for open source content management systems. Its central focus is shifting from building websites, where desktop users came first, to supporting a whole range of ways in which content is now consumed on the Internet.

Drupal 8 comes with mobile-first, responsive themes by default. HTML5 is baked in, making Drupal content viewable by any user with a device that supports a web browser. RESTful web services are included in Drupal 8 core, meaning that Drupal content can be consumed through apps or other machines, as well as through the traditional web browser.

Unsurprisingly, major changes in the way Drupal is used are accompanied by some major changes in the way you will be working with it as a developer. This book will guide you through the exciting and far-reaching changes that Drupal 8 brings. You'll learn how to build complex, powerful web applications by configuring Drupal without having to write any code. You'll also learn how to create responsive, mobile-first themes, write custom modules, and manage your Drupal projects using modern incremental development techniques.

If you're already a PHP developer, Drupal 8 is going to be an exhilarating ride for you. Modern object-orientated programming techniques that use many elements of the Symfony PHP framework are going to help you write more flexible, robust, and reusable code. The Drupal community will be attracting more and more non-Drupal PHP programmers over the coming years, as it's more elegant and cutting edge than before. Whether or not you're part of the Drupal community, there has never been a better, more exciting time to get heavily involved.

## What this book covers

*Chapter 1, Setting Up a Drupal Development Environment*, walks you through the setup of a Drupal development environment as the professionals do it. You'll be introduced to a range of powerful tools that will make Drupal development more efficient and fun, and it will also help you drastically improve your productivity.

*Chapter 2, Custom Module Development*, gets stuck into code. You'll be writing object-orientated code to make your modules more flexible and extendable. The chapter will also cover test-driven development, which is the basis of everything you need to deploy Drupal sites that consistently do what they should.

*Chapter 3, Drupal Views and Configuration Management*, showcases the new configuration management system of Drupal 8, which allows you to properly separate site configuration from site content. You'll see how content types and field configuration is now done using a markup language called YAML.

*Chapter 4, Introduction to the Field Types API and Developing the Custom Field Module*, explores how HTML5 support has been built into Drupal 8 and the new features that this provides. You'll be building a new custom field to showcase some of these features.

*Chapter 5, Theming in Drupal 8*, introduces the Twig templating engine and guides you through building a fully responsive theme. The chapter will also cover theme hooks that allow you to further modify the default look and feel of Drupal 8.

*Chapter 6, Enhancing the Content Author's User Experience*, illustrates how Drupal 8 vastly improves the administration interface, where users have often been sadly forgotten in previous versions of Drupal. Drupal 8 isn't just for end users, it's also for editors, moderators, and administrators of the site too.

*Chapter 7, Adding Media to Our Site*, helps you launch a multimedia campaign by introducing a range of media elements to your site. You'll learn how to make your site more compelling and convey the required message, not only in words, but also in other types of media.

*Chapter 8, How Does it Taste? – Getting Feedback*, walks you through the built-in functionality of Drupal 8 to get feedback from your users. This chapter will also cover starting a two-way conversation by building a form to help your users communicate with you and submit their views or requests.

*Chapter 9, Advanced Views Development*, takes an in-depth look at the Views module, which you'll use in many of your Drupal projects. We'll look at how to create a custom Views plugin that uses jQuery.

*Chapter 10, Drupal Project Management and Collaboration*, introduces you to a place where you can make some best friends—the Drupal community. We'll look at tools for collaboration, how to get more involved in the Drupal community, and how to work as a team to build something amazing.

*Chapter 11, Searching Your Site with the Search API Module*, introduces some powerful search functionality provided by the Search API, a framework for extending Drupal's standard search. We'll look at what it can achieve on its own, and then take a look at Apache Solr, a third-party search engine that will make your search functionality lightning fast.

*Chapter 12, RESTful Web Services in Drupal*, breaks down all the boundaries of your Drupal development. You'll create a custom API using REST and an AngularJS app that will consume content from Drupal using the API. Now Drupal doesn't just need to power websites—it could power almost anything.

*Appendix, Pop Quiz Answers*, covers all the answers enlisted in the pop quiz sections of the book.

## What you need for this book

To follow the examples in this book, you'll need a computer on which you can set up your own development environment with a number of useful tools. All these tools are either open source or have free equivalents. The PhpStorm IDE mentioned in *Chapter 1, Setting Up a Drupal Development Environment*, is a proprietary and fairly expensive piece of software, but you could also use NetBeans or Eclipse as alternative IDEs, as they have comparable features.

You're also going to need an Internet connection. Many of the examples in this book rely on external services such as GitHub or PuPHPet and you'll have to do some fairly large downloads if you're installing some of the recommended software packages from scratch.

## Who this book is for

This book is aimed at people who would like to start configuring, developing, or theming with Drupal 8. Although no experience with previous versions of Drupal is necessary, you'll find many familiar Drupal concepts if you've already used Drupal 7.

For development-focused chapters, it's assumed that you have some prior knowledge of PHP and are aware of modern PHP development practices. The theming chapters will assume that you are familiar with HTML and CSS. Some experience using the command line would be useful, but not essential.

Above all, an inquisitive mind, a readiness to make mistakes, and an enthusiasm to embark on a great Drupal adventure are most important.

## Sections

In this book, you will find several headings that appear frequently (Time for action, What just happened?, Pop quiz, and Have a go hero).

To give clear instructions on how to complete a procedure or task, we use these sections as follows:

### Time for action – heading

- 1.** Action 1
- 2.** Action 2
- 3.** Action 3

Instructions often need some extra explanation to ensure they make sense, so they are followed with these sections:

### **What just happened?**

This section explains the working of the tasks or instructions that you have just completed.

You will also find some other learning aids in the book, for example:

### Pop quiz – heading

These are short multiple-choice questions intended to help you test your own understanding.

### Have a go hero – heading

These are practical challenges that give you ideas to experiment with what you have learned.

## Conventions

You will also find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Navigate to the module page on Drupal to get the link to the latest packaged `tar.gz` file."

A block of code is set as follows:

```
<div>
  <ul>
    <li>content 1</li>
    <li>content 2</li>
  </ul>
</div>
```

Any command-line input or output is written as follows:

```
git checkout [filename]
git reset --hard
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Click on **Save** and then re-index your content."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for this book from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the book's webpage at the Packt Publishing website. This page can be accessed by entering the book's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- ◆ WinRAR / 7-Zip for Windows
- ◆ Zipeg / iZip / UnRarX for Mac
- ◆ 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/Drupal-8-Development-Beginners-Guide-Second-Edition>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

## **Piracy**

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

## **Questions**

If you have a problem with any aspect of this book, you can contact us at [questions@packtpub.com](mailto:questions@packtpub.com), and we will do our best to address the problem.



# 1

## Setting Up a Drupal Development Environment

*In this chapter, we will be setting up a development environment for the Drupal framework. We will be also installing and exploring Drupal 8 from a developer's perspective and learn about the virtual development environment.*

In this chapter, we will cover:

- ◆ Installing Drupal for local development
- ◆ Setting up PHPStorm IDE for Drupal development
- ◆ Installing, configuring, and using Drush 7 with Drupal 8
- ◆ Installing Drupal 8
- ◆ GitHub for Drupal development
- ◆ Installing Vagrant using PuPHPet

### Installing Drupal for local development

You can install Drupal using different web servers and databases. The most commonly used combination is Apache, MySQL, and PHP, often referred as the \*AMP stack.

Specific to Drupal, there are two main ways you can do this:

- ◆ Acquia Dev Desktop (for Mac and Windows only)
- ◆ AMP: Manual installation of Apache MySQL PHP

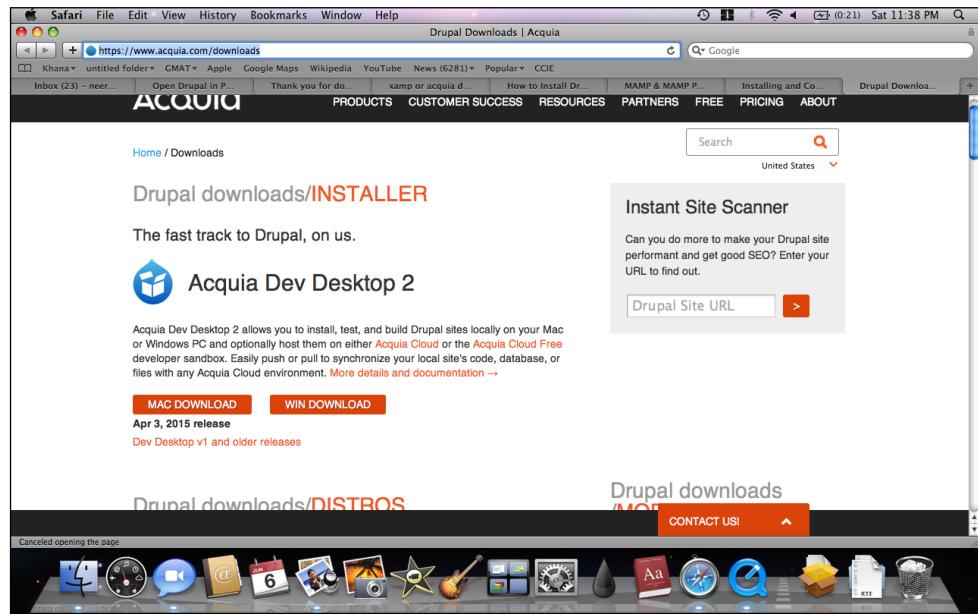
These two ways have the following minimum system requirements:

- ◆ **Disk space:** A minimum installation requires 15 megabytes. 60 MB is needed for a website with many contributed modules and themes installed.
- ◆ **Web server:** Apache, Nginx, Microsoft IIS, or any other web server with proper PHP support.
- ◆ **Database:**
  - ❑ MySQL 5.5.3/MariaDB 5.5.20/Percona Server 5.5.8 or higher with PDO and an InnoDB-compatible primary storage engine
  - ❑ PostgreSQL 9.1.2 or higher with PDO
  - ❑ SQLite 3.6.8 or higher

## Time for action – installing Drupal using Acquia Dev Desktop

If you are using Mac or Windows, Acquia Dev Desktop is the easiest method to develop Drupal:

1. Visit <https://www.acquia.com/downloads> and download the installer you need.

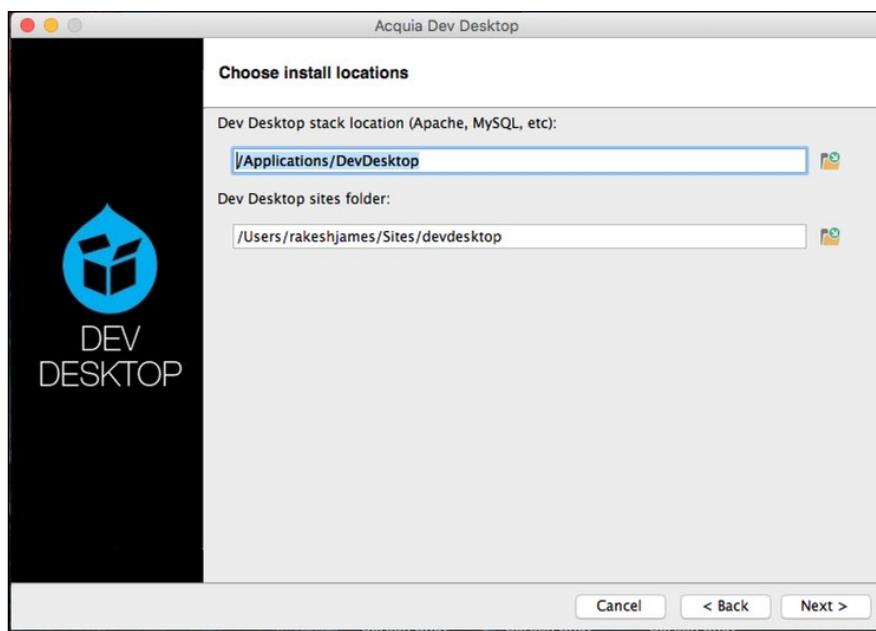


2. Open the Acquia Dev Desktop file and start the installation process. If you are a Windows user, find the folder that you used to save your Acquia Dev Desktop download and double-click on the .exe installation file.

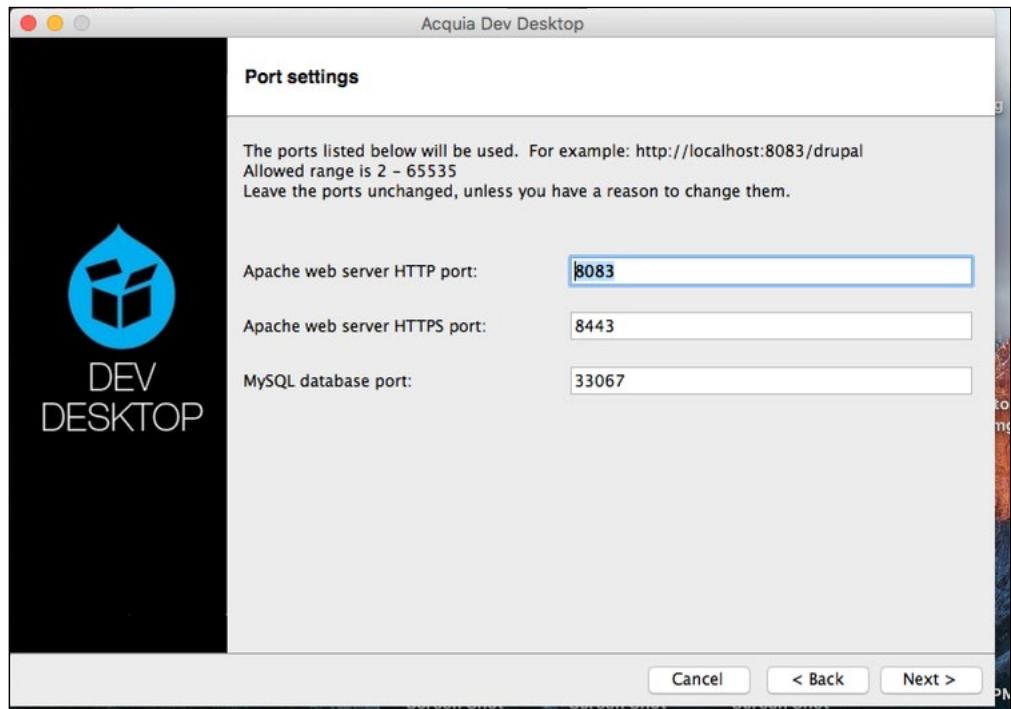
3. If you are a Mac user, find the folder that you used to save your Acquia Dev Desktop download. Move the file to the Application folder and double-click on the .dmg installation file. A new folder will pop up, with an icon that says **Acquia Dev Desktop Stack Installer**. Go ahead and click on that. Allow the installer to run on your Mac or PC.



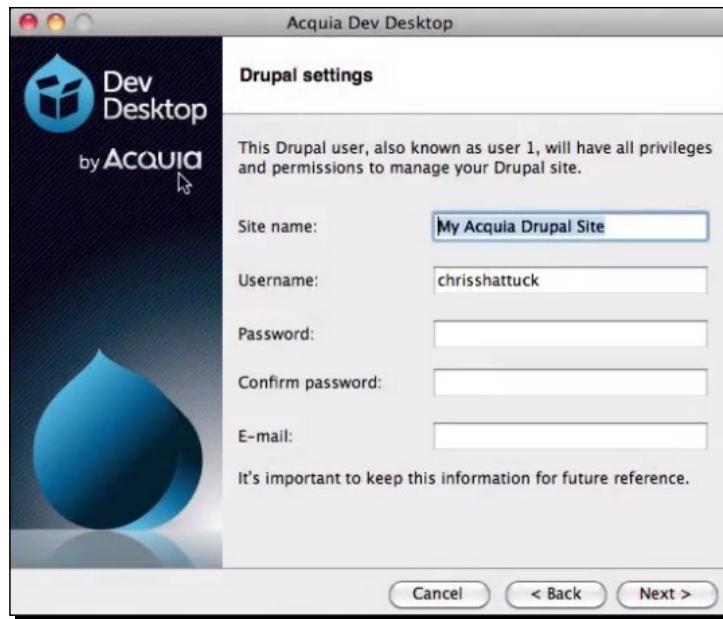
4. Let the installer run and click on **Next** for every step once you have read and agreed to the terms and conditions. Next, you can select where you are going to install the application stack (Apache, MySQL, and PHP) and where you are going to install your first Drupal site.



5. During this installation, we won't use our Drupal site directly but in other chapters, if you decide to use Dev Desktop for development work, you can import sites from other locations on your hard drive as well. Leave these at the default and click on **Next**. On the next page, you have to select the port number for Apache and MySQL. In most cases, it will work without any conflict, but if you are using multiple AMP stacks, you can go ahead and make the changes.



6. On the next screen, we set the defaults for our first Drupal site. Fill this out and click on **Next**. Make sure you note down the username and password. You will need this to log in to your Drupal site.



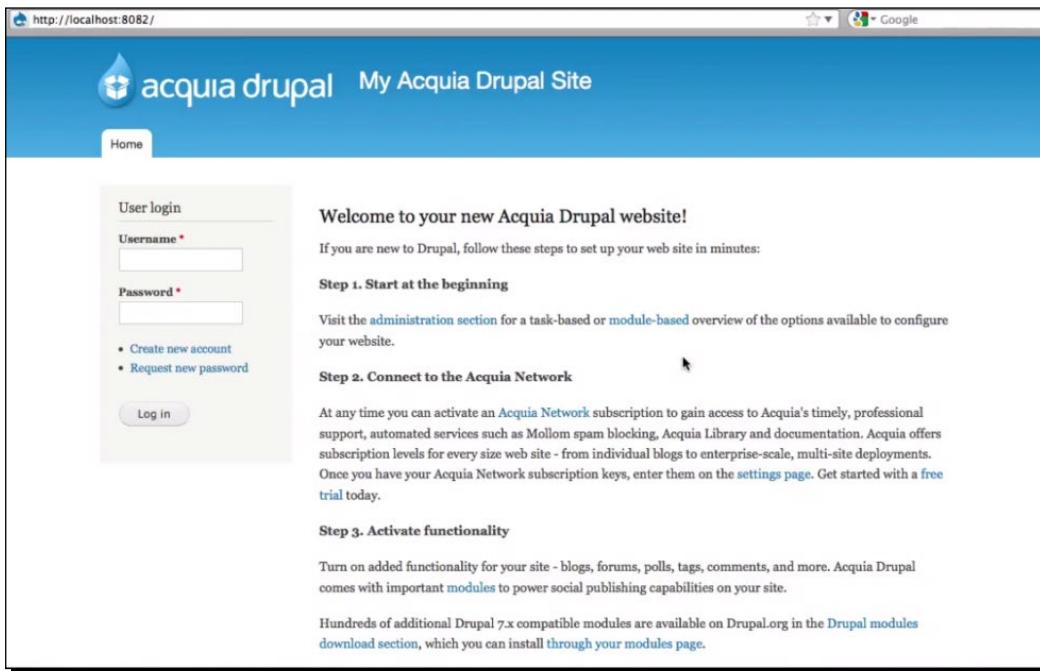
Creating credentials for the first Drupal site

7. Click on **Next** to begin the installation process. Once the installation is complete, click on **Finish**, and this will open up the **Acquia Dev Desktop Control Panel**.



Acquia Dev Desktop Control Panel

- 8.** Go to your new site by clicking on **Go to my site**. Use **Manage my database** to browse to **PHPMyAdmin**.



Login page for setting up the site

## ***What just happened?***

You have installed Drupal using Acquia Dev Desktop. From here, you can begin working on your new Drupal site.

Note that the site is running using Acquia Drupal, which is little different in that you would download from your <https://www.drupal.org/>. Installing Acquia Dev Desktop on Windows is almost similar.

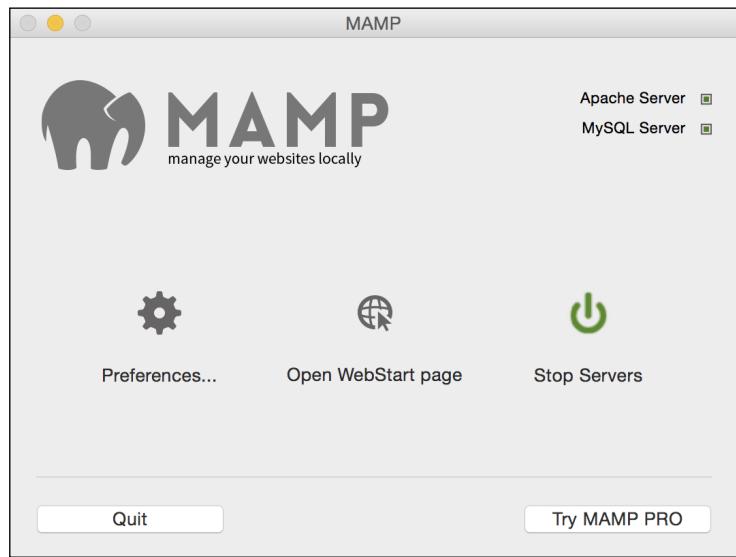
## **Installing Drupal the localhost way**

Now we will go through installation steps of AMP packages for both Mac OS X and Windows operating systems.

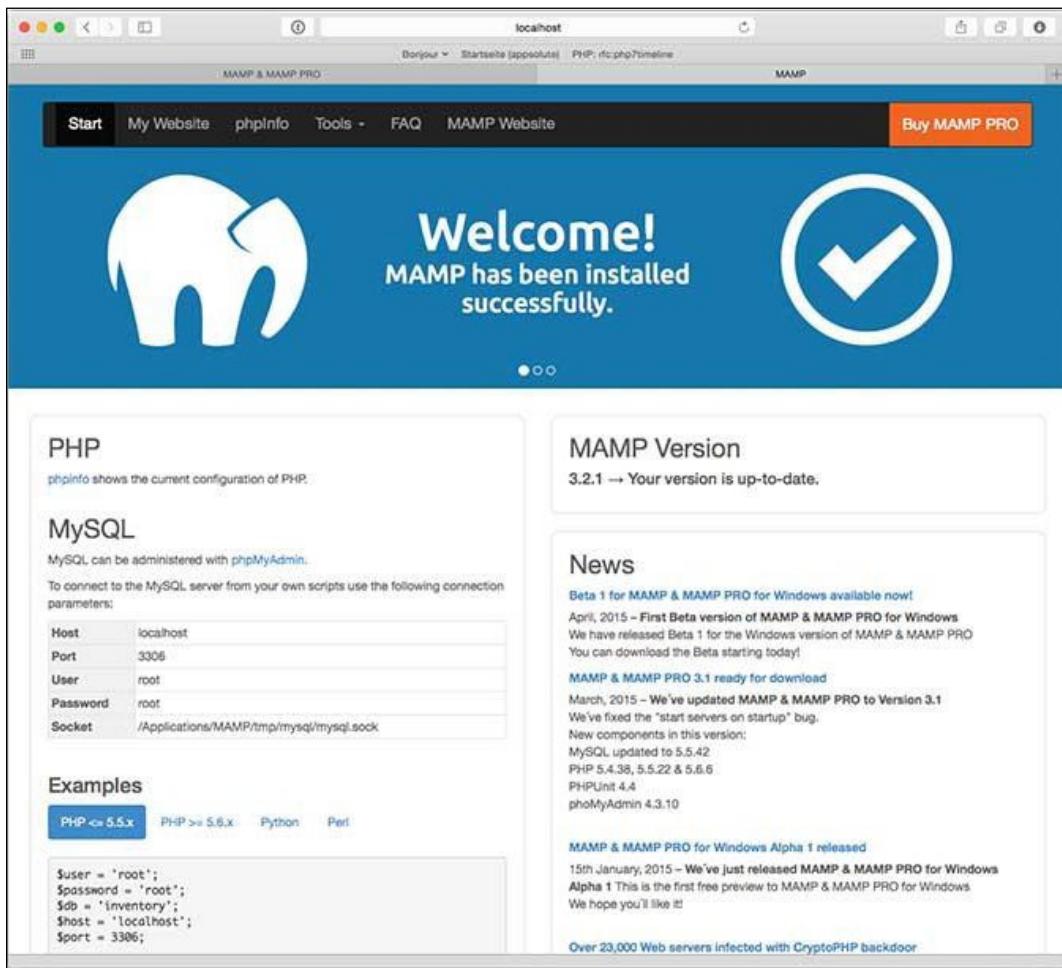
### **Time for action – installing a Mac OS X AMP stack**

For Mac OS X, we will use the MAMP package to install an AMP stack.

- 1.** First, download the latest version of MAMP from <https://www.mamp.info/en>. Once MAMP is completed downloading, double-click on the downloaded .pkg file. Move the file to the application folder and double-click on the MAMP .pkg file. This will launch the MAMP installer. The system installer will guide you through the installation process.
- 2.** After completing the installation process, launch your local server.



- 3.** Start MAMP and click on the **Start Servers** button. You will see the server status in the top-right corner. Click on the **Open WebStart page** button. You will see the default MAMP start page having links to access utilities such as phpMyAdmin, phpInfo, SQLite Manager, phpLiteAdmin, FAQ, MyFavoriteLink, and the MAMP website.



The default MAMP start page

4. After successful installation, you can launch your local servers. Start MAMP and click on the **Start Servers** button. In the status display in the top-right corner, the launch status of the servers is displayed. If necessary, you will be asked for your administrator password.



The process for setting up a development environment for Mac OS X is very similar to setting a development environment for a Linux distribution. They are both Unix-based operating systems. If you aren't already tied to a particular Linux distribution and would like to set up a development environment in Linux, then I highly recommend Ubuntu. There are excellent directions on setting up a Drupal development environment available at <https://help.ubuntu.com/community/Drupal>.

### ***What just happened?***

Congratulations! You now have a working AMP stack installed on your Mac.

## **Time for action – installing a Windows AMP Stack**

For windows, we will use XAMP:



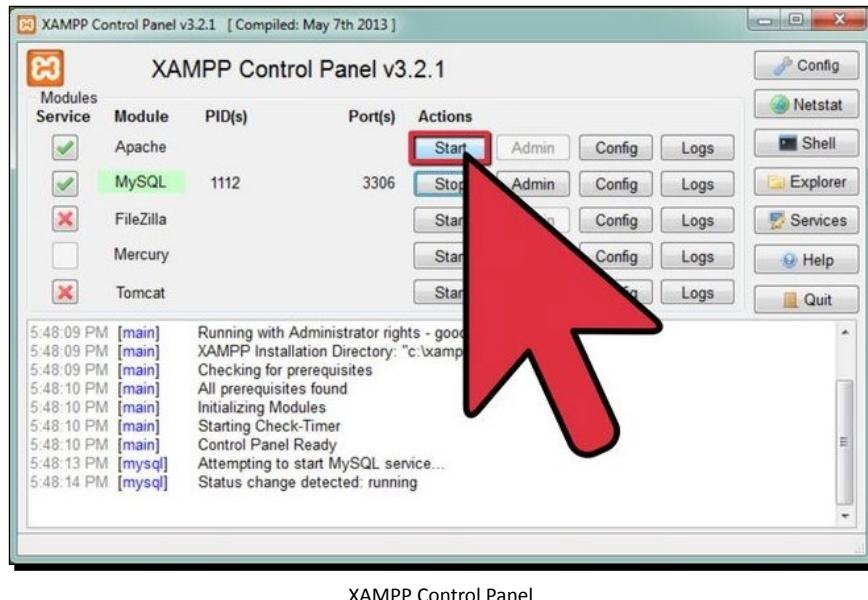
I do most of my Drupal development on Mac OS or Ubuntu. Unix-based OS is a better fit for Drupal development as there are many development-oriented aspects of Drupal that are Unix-centric. From cron to .htaccess based clean URL, a lot of documentation on <http://drupal.org/> will be biased towards the Unix OS.

1. Download the latest version of XAMPP from <https://www.apachefriends.org/download.html>. Once your download is complete, double-click on the .exe file to install the program. Accept the default settings and complete the installation process. You can change the settings by editing the configuration files later at any point in time.

*Setting Up a Drupal Development Environment*

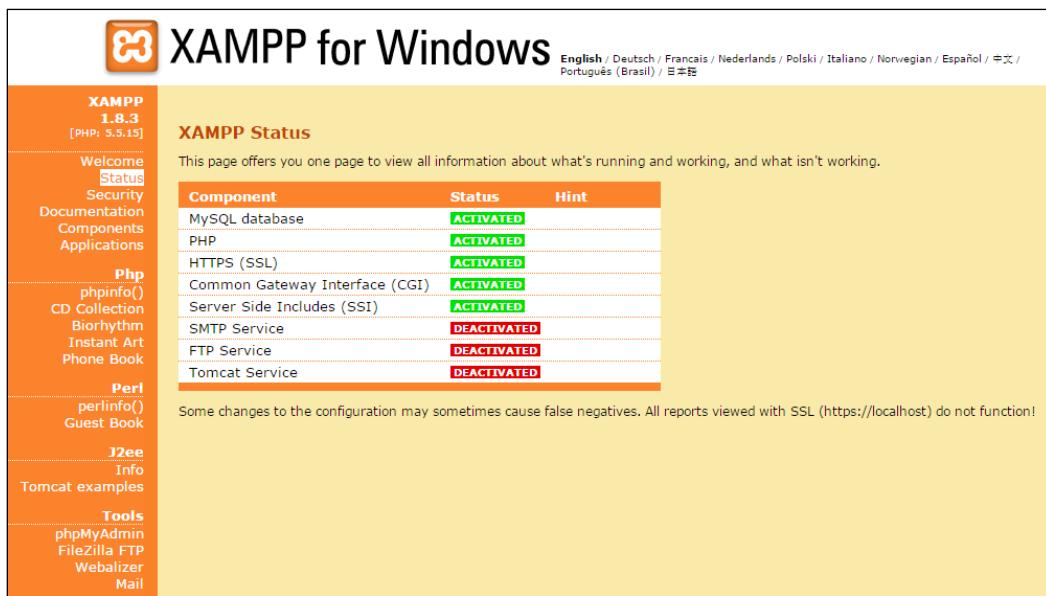
---

- 2.** After completing the installation process, start the XAMPP control panel.



XAMPP Control Panel

- 3.** Inside the XAMPP Control Panel application, click on the **Start** button next to Apache and MySQL. Now open up your favorite web browser, navigate to `http://localhost`, and you should see something similar to the following screenshot:



4. XAMPP is not meant for production use but only for development environments. Secure XAMPP before publishing anything online. You should go to the URL `http://localhost/security/`. With the security console, you can set a password for the MySQL user `root` and `phpMyAdmin`.

## ***What just happened?***

Congratulations! You now have a working AMP stack installed on your Windows PC. And also you have a working AMP stack installed on your Mac!

## **PHP configuration**

Drupal 8 recommends PHP version 5.4 or higher with the CURL extension. The latest version of MAMP includes PHP version 5.5 (it also includes an older PHP version and allows you to switch between them). The latest version of XAMPP for Windows includes PHP version 5. Although this version of PHP meets the requirements of Drupal 7, there are some PHP-related settings that need to be tweaked before we install Drupal in order to ensure that things will run smoothly. The PHP requirements list is from Drupal 8 at <https://www.drupal.org/requirements/php>.

## **Time for action – modifying the php.ini settings**

To modify the `php.ini` settings in Mac OS X and Windows:

- ◆ **Mac OS X:** Use your favorite text editor to open the `php.ini` file located at `/Applications/MAMP/bin/php/php5.4.6/conf`.
- ◆ **Windows:** Use your favorite text editor to open the `php.ini` file located at `C:\xampp\php`. I like Notepad++.

Navigate to the respective folder location in Mac OS X and Windows, and edit the settings to match the following values:

```
max_execution_time = 60;
max_input_time = 120;
memory_limit = 128M;
error_reporting = E_ALL & ~E_NOTICE
```

## ***What just happened?***

Congratulations! You have successfully modified the `php.ini` file in Mac OS X and Windows systems.

## Modifying the MySQL my.cnf settings

To modify the MySQL `my.cnf` settings in Mac OS X and Windows:

- ◆ **Mac OS X:** MAMP does not use a `my.cnf` file by default. So you must copy the file at `/Applications/MAMP/Library/support-files/my-medium.cnf` to `/Applications/MAMP/conf/my.cnf` (notice the new name of the file).
- ◆ **Windows:** For XAMMP, open the `my.ini` file located at `C:\xampp\mysql\bin`.



You may want to make a backup copy of this file before you begin to edit it.



### Time for action – setting up MySQL for Drupal

Open the `my.cnf/my.ini` file in your text editor, and find and edit the following settings to match these values:

```
# * Fine Tuning
#
key_buffer = 16M
key_buffer_size = 32M
max_allowed_packet = 16M
thread_stack = 512K
thread_cache_size = 8
max_connections = 300
```

One of the real goals for the Drupal MySQL configuration is the `max_allowed_packet` setting. This has always been a source of bewildering errors in the past for me and many other Drupal developers that I know, and it is a setting that is specifically mentioned on the <http://drupal.org/requirements#database> page, under the database server section.

When you are ready to make your site live, there are some excellent performance tuning tips available at <http://drupal.org/node/2601>.

### **What just happened?**

Congratulations! You have successfully updated the MySQL settings in Mac OS X and Windows systems.

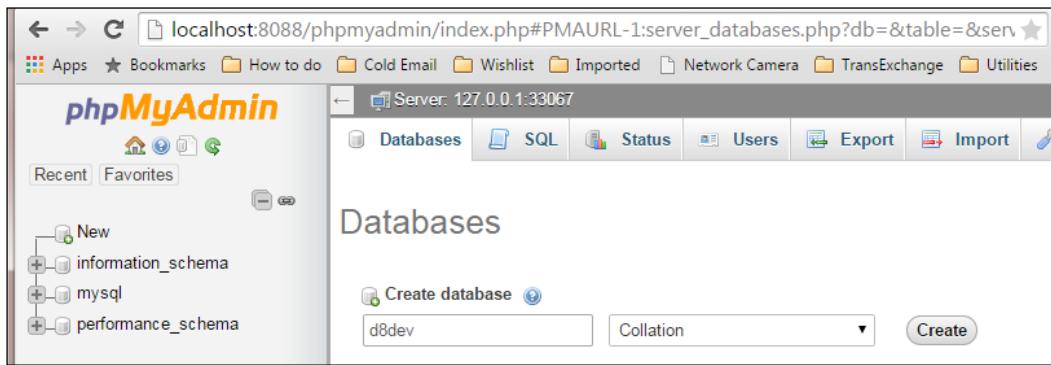
## Time for action – creating an empty MySQL database

Before we can install Drupal, we need to create a new and empty MySQL database.

Both MAMP and XAMPP include phpMyAdmin—a web-based administration tool for MySQL. We will use phpMyAdmin to create an empty database for Drupal.

- ◆ **Mac OS X:** With MAMP running, open your favorite web browser and go to <http://localhost:8888/phpMyAdmin>.
- ◆ **Windows:** With XAMPP running, open your favorite web browser and go to <http://localhost/phpmyadmin/>.

You will see the following screen:



### **What just happened?**

You have installed a fully functional AMP stack that has been configured specifically for Drupal, and you have created an empty MySQL database as a preliminary step for installing Drupal.

## Installing Git

Git is a free source control and versioning software that has become very popular over the last few years. In February 2011, Drupal (<https://www.drupal.org/>) migrated from the outdated CVS versioning system to Git. The migration to Git has enabled a completely new way for Drupal developers to interact with Drupal (<https://www.drupal.org/>), and we will highlight this enhanced interaction throughout the book. However, we will also immediately start using Git to facilitate setting up a Drupal development environment. So, if you don't have Git already installed on your computer, let's get it set up.

## Time for action – installing Git for Mac OS X

To install Drush for the Mac, we are going to use Homebrew (an open source package manager for Mac OS X), with installation instructions available at <http://brew.sh/>.

Once you have Homebrew installed, installing Git is as easy as opening up the Terminal application (in /Applications/Utilities) and typing the following command:

```
brew install git
```

## Time for action – installing Git for Windows

Download the installer for Windows from <http://git-scm.com/downloads>. Double click on the .exe file to install the program.

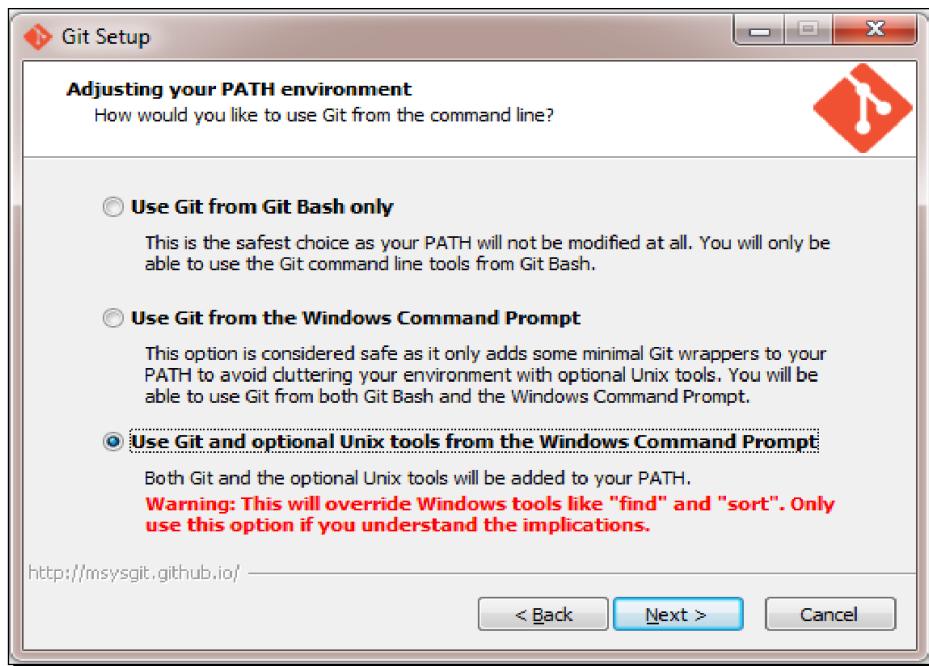
1. Click on the downloaded file link to begin the installation process.



The Git setup wizard

2. Click on **Next**, and then on **Next** again on the **GNU General Public License** screen.
3. On the **Select Destination Location** screen, click on **Next** to accept the default destination.
4. On the **Select Components** screen, accept the defaults again and click on **Next**.

5. On the **Adjusting your PATH Environment** screen, select **Use Git and optional Unix tools from the Windows Command Prompt** as this will allow Git to work with Drush, which we will cover next.



Choosing how to use Git from command line

## ***What just happened?***

You have installed the Git version control system—a tool that will greatly facilitate interaction with the existing contributed code at Drupal (<https://www.drupal.org/>).

## **Installing Drush**

Drush, a portmanteau of the words Drupal and shell, is a command-line utility that facilitates the management of a Drupal environment from your favorite shell (the Terminal application on Mac OS X and Ubuntu, and the Command Prompt application on Windows). For example, installing a contributed module on Drupal (<https://www.drupal.org/>) can be as easy as running the following commands from the command line:

```
drush dl modulename  
drush en modulename -y
```

## Time for action – installing Drush for Mac OS X

To install Drush for Mac, we are going to use Homebrew again at <https://www.drupal.org/node/954766>.

In the Terminal, type the following command:

```
brew install drush
```

## Time for action – installing Drush for Windows

To install Drush in Windows, we will need to use Cygwin that provides a Linux-like environment for Windows.

We are now going to install Drush using Composer in Cygwin:

1. Download and install Cygwin
2. In the packages section install the following packages:
  - ncurses: This is a library used for showing text-based user interface.
  - git: This is a version control package
  - bsdtar: This is used to untar compressed tar files.
  - curl: This is a client-side URL transfer package.
3. Now installer Composer, a PHP dependency package management tool, using the Composer installer.
4. From Cygwin terminal, type the following command to install Drush 8:  
`composer global require drush/drush:8.*`



Learn more about Cygwin project at <http://www.cygwin.com/>.  
For more information on installing Drush, check Install Drush on Windows—the easy way at <https://www.drupal.org/node/594744>

## Installing Drupal 8

All right, now we are getting somewhere. Now that we have created a database, installed Git, and installed Drush, we have everything in place to install Drupal 8.

## Time for action – installing Drupal 8

We are going to use a combination of Drush and Git to install Drupal:

1. **Mac OS X:** Open up the Terminal application, and type the following command:

```
cd /Applications/MAMP/htdocs
```

**Windows:** Open Command Prompt and type the following command:

```
cd C:\xampp\htdocs
```

2. Now, we are going to use Git to locally clone the Drupal core Git repository into a new d8dev folder (this will take a few minutes or so depending on your network bandwidth):

```
$ git clone --branch 8.1.x http://git.drupal.org/project/drupal.  
git d8dev  
Cloning into 'd8dev'...  
remote: Counting objects: 456756, done.  
remote: Compressing objects: 100% (95412/95412), done.  
remote: Total 456756 (delta 325729), reused 445242 (delta 316775)  
Receiving objects: 100% (456756/456756), 100.04 MiB | 8.60 MiB/s,  
done.  
Resolving deltas: 100% (325729/325729), done.  
cd d8dev
```

3. Next, we want to use Git to switch to the latest Drupal 8 release. First, we will list all the available releases:

```
8.0-alpha1  
8.0-alpha10  
8.0-alpha11  
8.0-alpha12  
8.0-alpha13  
8.0-alpha2  
8.0-alpha3  
8.0-alpha4  
8.0-alpha5  
8.0-alpha6  
8.0-alpha7  
8.0-alpha8  
8.0-alpha9  
8.0.0  
8.0.0-alpha14  
8.0.0-alpha15
```

```
8.0.0-beta1  
8.0.0-beta10  
8.0.0-beta11  
8.0.0-beta12  
8.0.0-beta13  
8.0.0-beta14  
8.0.0-beta15  
8.0.0-beta16  
8.0.0-beta2  
8.0.0-beta3  
8.0.0-beta4  
8.0.0-beta5  
8.0.0-beta6  
8.0.0-beta7  
8.0.0-beta8  
8.0.0-beta9  
8.0.0-rc1  
8.0.0-rc2  
8.0.0-rc3  
8.0.0-rc4  
8.0.1  
8.0.2  
8.0.3  
8.0.4  
8.0.5  
8.0.6  
8.1.0  
8.1.0-beta1  
8.1.0-beta2  
8.1.0-rc1  
8.1.1  
8.1.2  
8.1.3
```

4. You will see that the latest release is 8.1.3, but you should substitute whatever the latest release may be for you and use that in the following Git command:

```
git checkout 8.1.3
```

```
Note: checking out '8.1.3'.
```

```
You are in the detached HEAD state. You can look around, make experimental changes, and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.
```

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

```
git checkout -b new_branch_name
```

HEAD is now at 079a52b. Revert Issue #2457653 by Gábor Hojtsy:  
System.site langcode is both used as a file language code and a site language code.

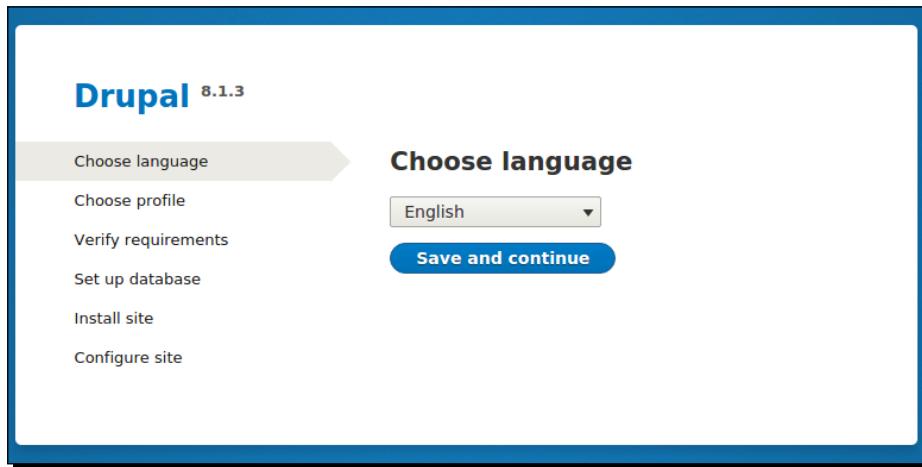
By using Git, we are linking our download of Drupal to Drupal's Git repository for our local core Drupal install. This will facilitate an easy update process for future Drupal core updates.

 Drupal 8 uses Symfony components, which is managed by using a vendor directory and has to be downloaded in the Drupal repository using Composer. For Drupal repositories cloned via Git, you need to run composer install, which will download the vendor directory.

Next, we are going to go through the web-based installation process to set up Drupal:

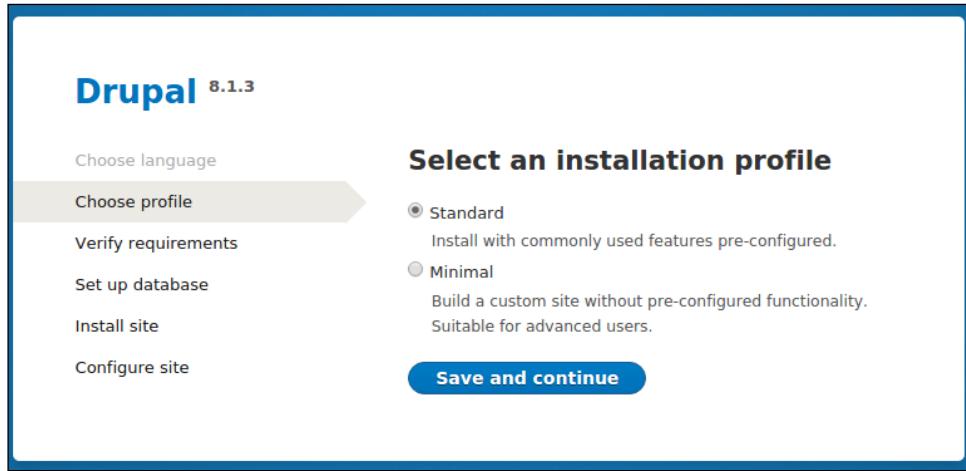
- **Mac OS X:** Open <http://localhost:8888/d8dev/> in your web browser
- **Windows:** Open <http://localhost/d8dev/> in your web browser

**5.** You should see the following screen:



Select your language and click on **Save and Continue**.

- 6.** Select the **Standard** profile and click on **Save and Continue**.



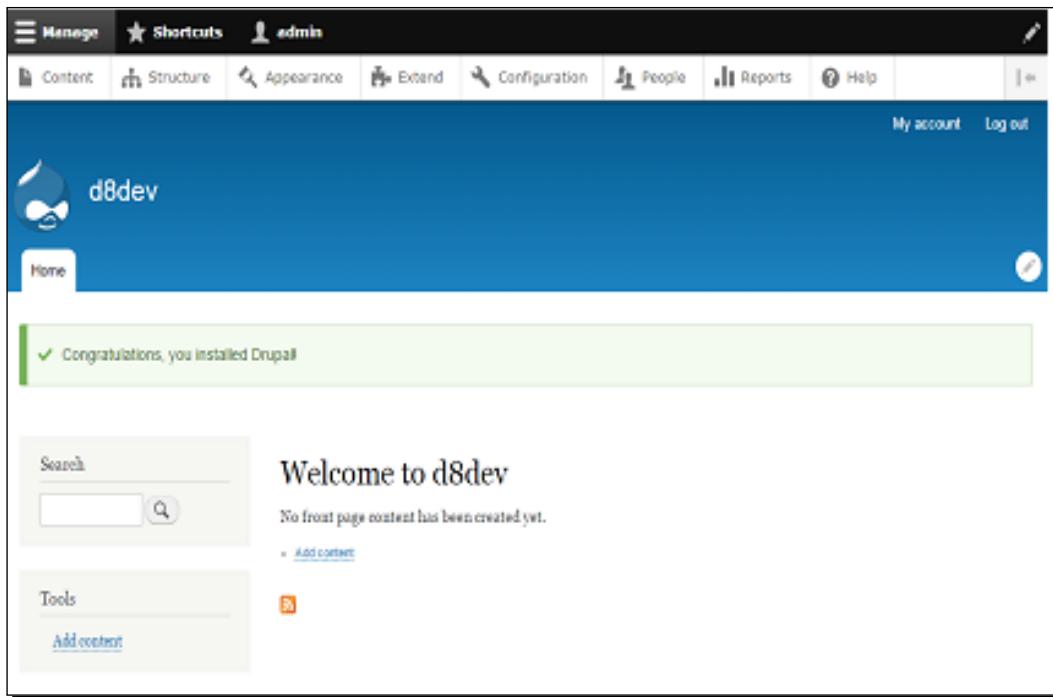
- 7.** On the **Database configuration** screen, select **MySQL**. Enter `d7dev` as the database name and `root` as the database username and password. Click on **Save and Continue**. You will see the site installation screen.

For a local development site, it is quite convenient to use the root MySQL user. However, for a live/production site, you should always create a unique MySQL user and password for your Drupal database.

- 8.** On the **Configure site** screen, enter the following values (the e-mail address doesn't have to be real but must appear valid):

- Site name:** d8dev
- Site e-mail address:** A valid e-mail address
- Username:** admin
- Password and Confirm password:** admin (this is only a development environment, so keep it simple)
- E-mail address:** The same as used for **Site e-mail address**

Fill out the rest of the form and click on **Save and continue**. Your Drupal site installation is complete. So click on the **Visit your new site** link to see the site.



## What just happened?

You used Git to get a brand new Drupal 8 site up and running. In the coming chapters, you will see how useful Git, along with Drush, can be for day-to-day Drupal development.

## Installing the PHPStorm IDE

PhpStorm has emerged as one of the most Drupal-friendly IDEs for developing and debugging Drupal modules and themes at <https://confluence.jetbrains.com/display/PhpStorm/Drupal+Development+using+PhpStorm>.

### Time for action – installing the PHPStorm IDE

Go to <https://www.jetbrains.com/phpstorm/download/>, and download the free 30-day trial of the correct version of the PHPStorm IDE for your operating system. Double-click (for Windows and Mac) on the file once it has completed downloading, and follow the instructions given on the page to install PHPStorm as per the documentation.

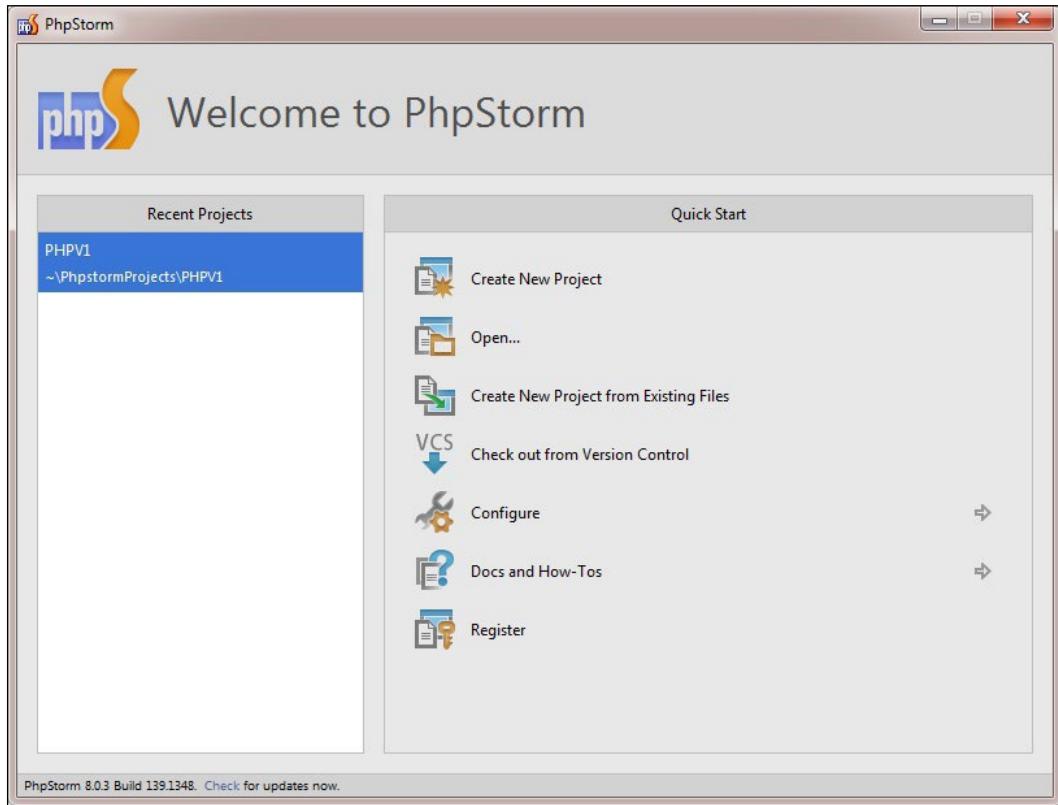


By no means should you feel like you have to use PHPStorm. There are a number of other good IDEs out there, and you may already be using a different IDE or may just be happy using your favorite text editor. However, I will be using PHPStorm throughout the book, so it may be easier to follow along if you are also using PHPStorm.

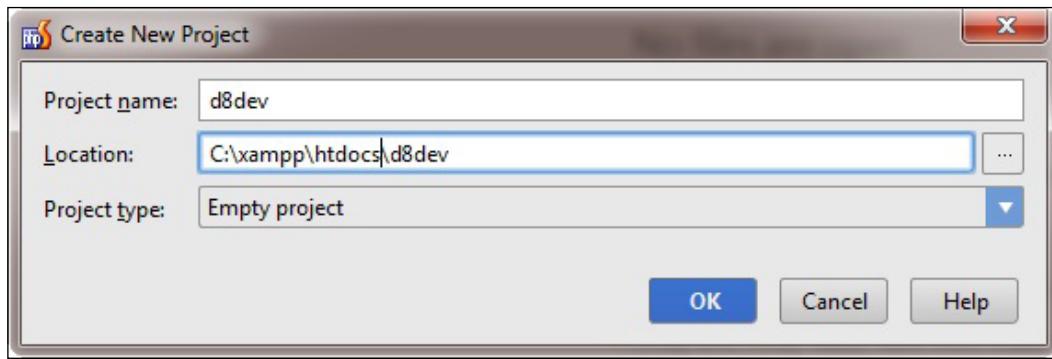
The Drupal development plugin is activated by default in PHPStorm. Otherwise, you can enable it in the settings dialog box.

## Time for action – creating a new PHPStorm project

Upon opening PHPStorm for the first time, you should see the following screen:



- Click on the **Create New Project** button, select **PHP Project**, and click on **Next**. At this point, you will see the **New PHP Project** window:



- Browse to your new Drupal 8 project. For Windows, it is `C:\xampp\htdocs\d7dev`, and for Mac OS X, it is `/Applications/MAMP/htdocs/d7dev`. After you have selected the location of your new Drupal 7 installation, click on **Finish**.

## Drupalize PHPStorm IDE

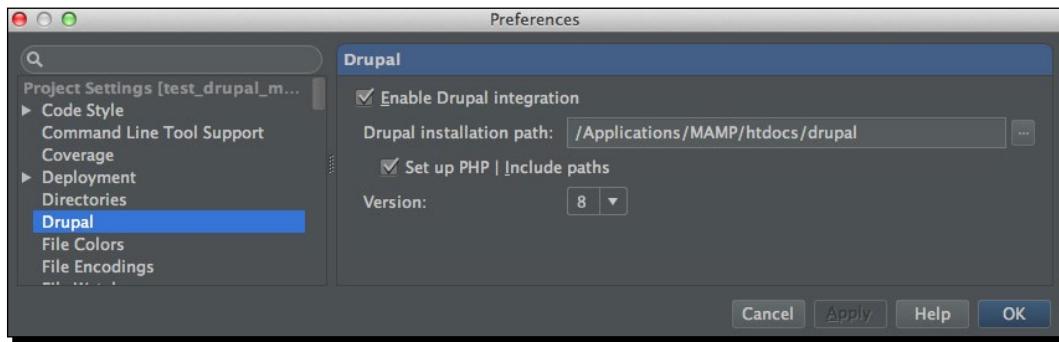
As soon as you set up a project in PHPStorm for an existing project or a new Drupal module, the IDE checks if the development environment is configured for Drupal or not. It will suggest you to enable Drupal support with a popup and an event in the Event log.



Clicking on the **Enable Drupal support** link will open the configuration dialog box. You can change the Drupal project installation path and select version number **8** for Drupal 8.



You can also change the Drupal integration settings later by opening the **Preferences** dialog.

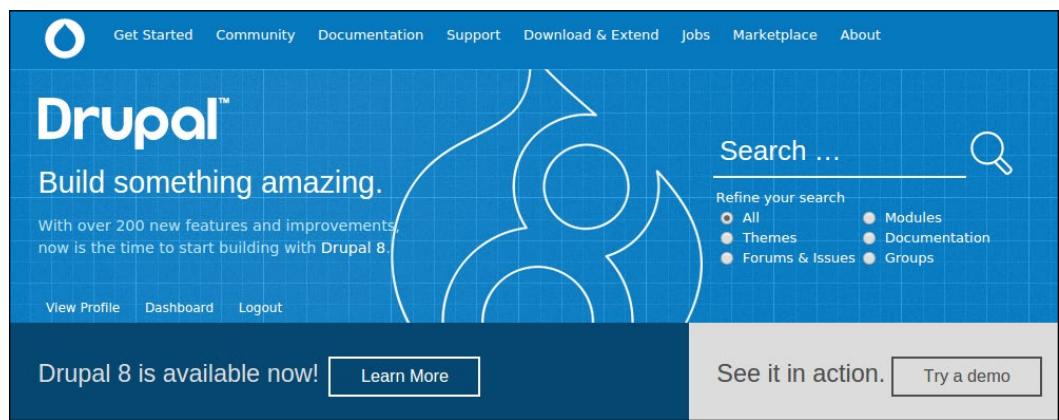


To get updated information about Drupal integrations with the latest version of PhpStorm, visit <https://confluence.jetbrains.com/display/PhpStorm/Drupal+Development+using+PhpStorm>.

## Drupal from a developer's perspective

If you are new to Drupal development, the first thing that you should understand is that being a good Drupal developer means being a part of the Drupal community. Drupal development is very much an open source process, and as such it is a community-driven process.

Drupal has numerous resources to assist developers of all experience levels. However, before you can take advantage of all of these resources, you need to be a member of Drupal (<https://www.drupal.org/>). So, if you haven't already joined Drupal, now is the time to do so at <http://drupal.org/user/register>. Once you are a member and have logged in to Drupal (<https://www.drupal.org/>), you will see the following screen:



As we move forward with Drupal 8 development, we will return to your Drupal (<https://www.drupal.org/>) dashboard on many occasions to keep a track of issues of the contributed modules that we are using. We will also utilize the new sandbox development feature that Drupal has added as part of the migration to Git.

## Installing Vagrant

As developers, a lot of times we need to set up a local version of the AMP stack that is similar to the production web server. Vagrant allows developers to build the virtualized environment once and keep sharing it for every case.

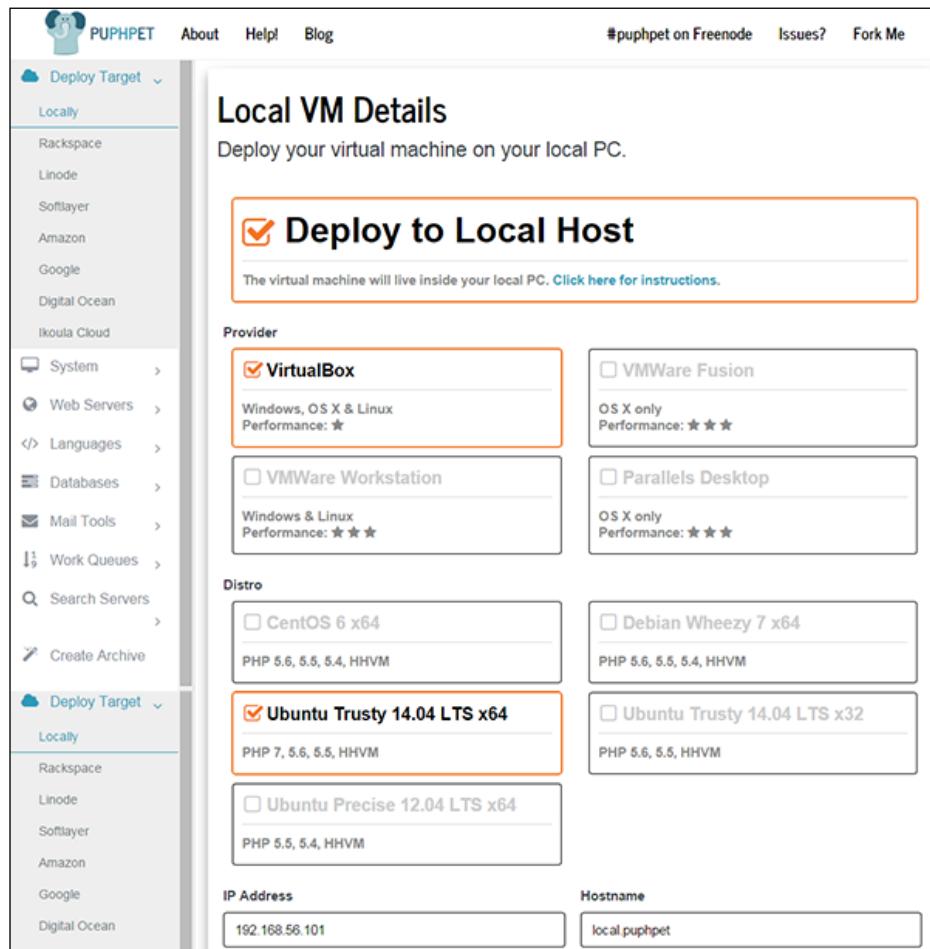
Vagrant is written in Ruby, so if you are unfamiliar with Ruby, writing Vagrant files could be challenging. This is where PuPHPet helps. Using PuPHPet, we can generate the package to create a virtual machine the same way as we specify on the PuPHPet website. We just have to provision it.

### Time for action – installing Vagrant

To install Vagrant, download the appropriate Version of vagrant packages and the VirtualBox for your system from <https://www.virtualbox.org/wiki/Downloads>:

- 1. Create a location:** Choose a location where you would like your VM to be saved. This location is regardless of where you save your VM configuration and files. Create a folder where you decide to store your VM. For example, in Windows, we can use C:/drupalvm.
- 2. Install VirtualBox:** Download and install the latest version of VirtualBox from <https://www.virtualbox.org/wiki/Downloads>, depending on the OS package.
- 3. Install Vagrant:** Installing Vagrant is easy. Just download your OS-specific package from <https://www.vagrantup.com/downloads.html>. Follow the OS-specific normal installation process.
- 4. Create your VM at PuPHPet:** Until now, we have installed Vagrant and VirtualBox. Now let's create our VM configuration at PuPHPet. Open the URL <https://puphpet.com/>. Here we will select what we need from our VM.

**5. VM Settings:** On the first page, click on **Get started right away**. This should lead to the following page. Here I have selected the distro as **Ubuntu Trusty 14.04 LTS x64**. You can select your distro and other configurations depending on your server choice. You can leave other settings as default if you are not aware of them.



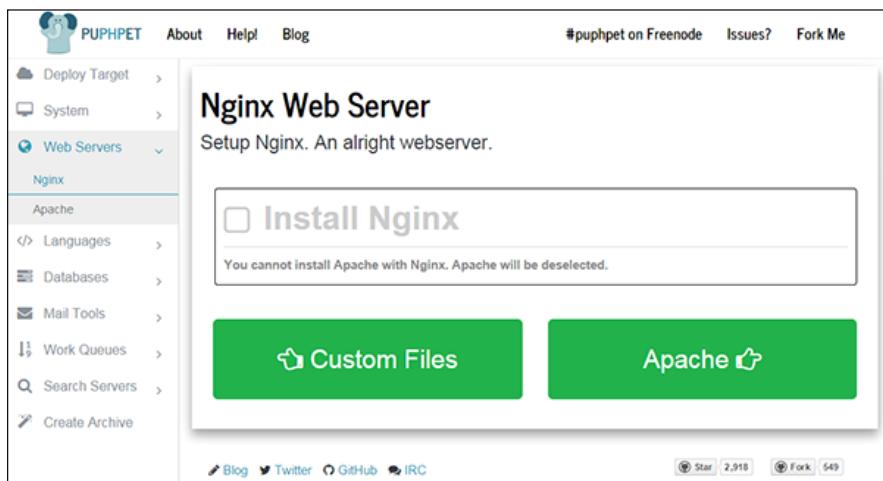
Selecting distro for your machine

Once you have selected your favorite package, hit the **System Packages** button.

In the next four steps, you can just leave the settings as they are and click on the buttons on the right-hand side to move forward.

You should get the **Nginx** option. Now I do not want Nginx as a web server. Instead I have opted for Apache.

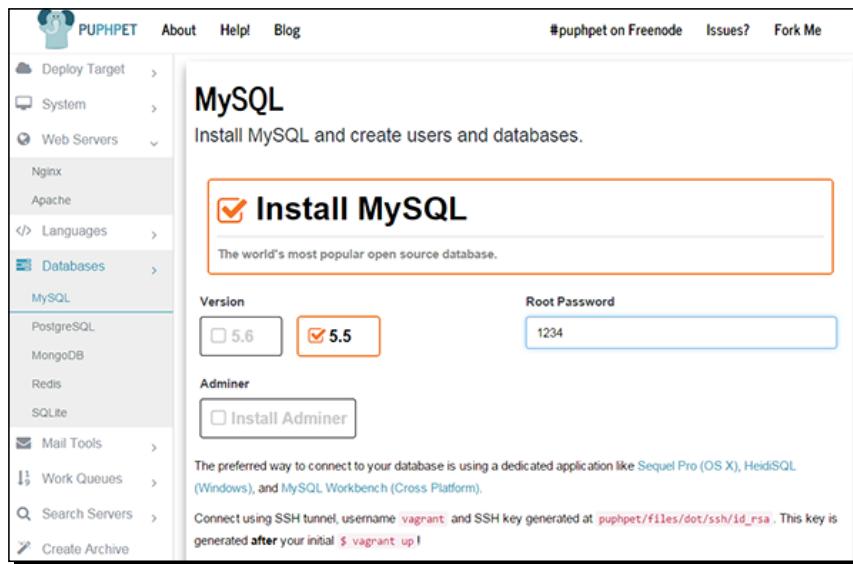
Just browse to the page's top and uncheck **Install Nginx**. This should collapse the Nginx configuration form and bring the button to move to the Apache configuration page.



Select the web server of your choice

You can select your web server configuration and move next to select the PHP configuration. Here, I have selected **Install Xdebug**, a good tool for Debugger, and a profiler tool for PHP.

After this, I just clicked on other language configuration without making any changes and moved on to MySQL.



Selecting database

Here you can select the MySQL version and root password, add more users, and create new databases. You can explore other databases too if you like.

Again, keep moving forward until you reach **Create Archive**, and you are done. Download your custom server config! This will download a ZIP file with the configuration of your custom VM.

1. **Set up your VM:** Extract the ZIP file you have downloaded just now and navigate to the location you just extracted your ZIP file to.

Open the Terminal in Mac OS X or Ubuntu. In Windows, you should open the command line with administrative privileges.

In the Terminal, browse to the folder location and type this:

```
vagrant up
```

Hit *Enter*. It will take a couple of minutes and you should see a message, everything finished installing.

2. **Setting up the host:** There is the last step of setting up the hostname before we start using our VM. For Drupal, it is always better to use proper domain names. To create virtual hosts, we need to make changes to the host files on our computer. Browse to the following folders depending on the OS:

- **Windows:** C:\windows\system32\drivers\etc\hosts
- **Linux (Ubuntu):** /etc/hosts
- **Mac:** /private/etc/hosts

You can use your favorite editor with administrative permission to make changes to the host file. I have selected to use drupal18.dev as my domain name. So let's add this line to the host file:

```
192.168.1.32 drupal18.dev www.drupal18.dev
```

We have used this IP address while creating our VM config file on PuPHPet. If you do not remember it now, you can find it by opening config.yaml in Notepad.

The config.yaml file is located inside your puphet folder, which you placed inside your VM folder earlier. You should see the IP address top next to private\_network.

We have just completed configuring a VM on our computer. You can open the browser and type your domain name—drupal18.dev.

## **What just happened?**

You have just installed Vagrant for project development. Now you can build a VM, which is your actual production server. And you can switch to different VMs depending on your different project requirements. What you need to install a new VM is a config file from PuPHPet.

## **Summary**

You have reached the end of the chapter! You should now have a working Drupal 8 website and PHPStorm IDE to begin custom Drupal development. We are yet to write actual code, so I am excited to get started with some development examples. However, this chapter has given us the tools and configuration system needed to make the development process much easier and enjoyable. Now, we are ready to begin developing for Drupal in earnest. In the next chapter, we will create a new content type and a basic custom module to change one of the fields on our content type.



# 2

## Custom Module Development

*We have setup our development environment in Chapter 1, Setting Up a Drupal Development Environment. Now let's get started and create a new content type and create a basic custom module to change one of the fields on our content type. We will also explore the concept of test driven development as good programming practice.*

In this chapter, we will learn about the following:

- ◆ Defining a custom content type (Recipe)
- ◆ Basics of object-oriented programming in Drupal 8
- ◆ Introduction of Symfony with Drupal 8
- ◆ Basics of Drupal 8 module development
- ◆ Field formatter API
- ◆ **Test-driven development (TDD)**
- ◆ Writing and running functional tests for custom module functionality using an inbuilt PHPUnit test framework
- ◆ Configuring and running unit tests with code coverage reports from within the PHPStorm IDE

## Creating custom Recipe content type

In the previous chapter, we installed Drupal 8 on our system. Now we are going to create our custom recipe content type. Before that, we will discuss five new fields added in Drupal 8:

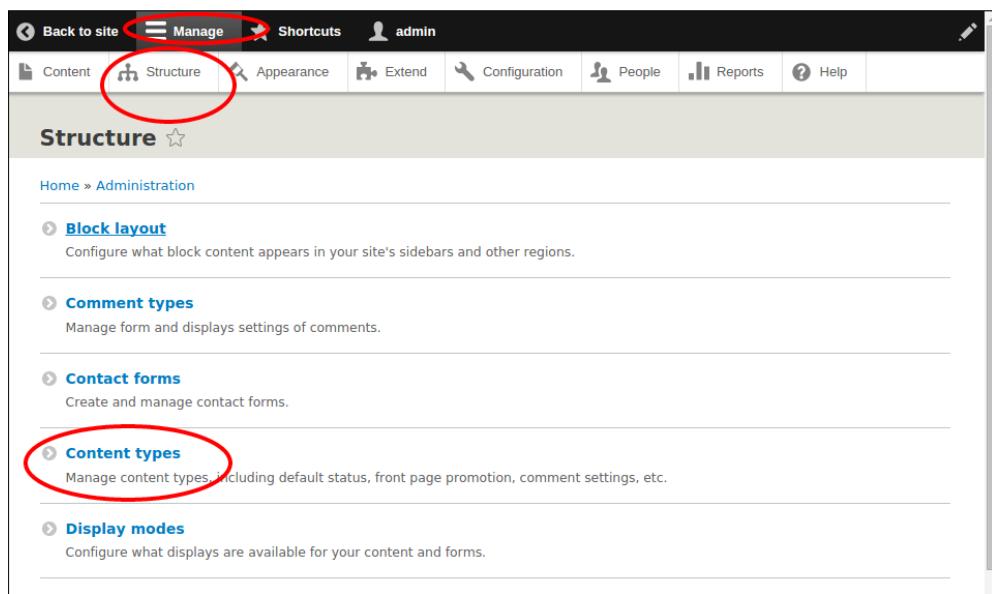
- ◆ **Date:** The **Date** field is the **Date** module in Drupal 7. We can choose to record the **Date and time** or **Date only** as options.
- ◆ **Email:** The **Date** module is simple but the **Email** field is even simpler. There are no settings at all for the **Email** field.
- ◆ **Link:** The **Link** field allows both internal and external links, along with a link text option.
- ◆ **Telephone:** The **Telephone** field is disabled by default in Drupal 8, so we need to enable the module to use it. It has no settings. It really is just a text field that adds `<a href="tel:...` to turn the text into a telephone link.
- ◆ **Reference:** The **Reference** module is the most powerful field. We can link to anything that's an entity, which means we can link to comments, content, blocks, files, terms, and users.

Now login to your Drupal 8 website. You should see a new admin toolbar.

### Time for action – creating custom content type

Let's create a custom content type by following these next steps:

1. Click on **Structure** in the admin toolbar, and then click on **Content types**.



2. On the **Content types** screen, click on the **Add content type** link.

The screenshot shows the 'Content types' page in a Drupal admin interface. At the top, there's a navigation bar with links like 'Back to site', 'Manage', 'Shortcuts', and 'admin'. Below that is a secondary navigation bar with 'Content', 'Structure', 'Appearance', 'Extend', 'Configuration', 'People', 'Reports', and 'Help'. The main content area is titled 'Content types' with a star icon. It shows two existing content types: 'Article' and 'Basic page'. The 'Article' type is described as 'Use articles for time-sensitive content like news, press releases or blog posts.' and has a 'Manage fields' button. The 'Basic page' type is described as 'Use basic pages for your static content, such as an 'About us' page.' and also has a 'Manage fields' button. A prominent blue button labeled '+ Add content type' is centered at the top of the list, with a red circle drawn around it to indicate it as the next step.

3. Write Recipe in the **Name** text field.
4. In the description, enter this text Simple recipe content type based on the schema.org base HTML5 Microdata schema for Recipes at: <http://schema.org/Recipe/>.
5. For the **Title** field label, enter name.

The screenshot shows the 'Add content type' form. At the top, it says 'Add content type' with a star icon. Below that is the breadcrumb trail: 'Home > Administration > Structure > Content types'. A note states: 'Individual content types can have different fields, behaviors, and permissions assigned to them.' The 'Name' field is filled with 'Recipe' (highlighted with a red circle), and the 'Machine name' is 'recipe [Edit]'. A note below says: 'The human-readable name of this content type. This text will be displayed as part of the list on the Add content page. This name must be unique.' The 'Description' field contains the text: 'A simple recipe content type based on the schema.org base HTML5 Microdata schema for Recipes at: <http://schema.org/Recipe/>'. Below the description is a note: 'Describe this content type. The text will be displayed on the Add content page.' On the left, there are several settings sections: 'Submission form settings' (with 'name'), 'Publishing options' (with 'Published, Promoted to front page'), 'Display settings' (with 'Display author and date information'), and 'Menu settings'. On the right, there are fields for 'Title field label' (set to 'name', highlighted with a red circle), 'Preview before submitting' (with 'Optional' selected), and 'Explanation or submission guidelines' (with a note: 'This text will be displayed at the top of the page when creating or editing content of this type'). At the bottom, a blue button labeled 'Save and manage fields' is highlighted with a red circle.

## Custom Module Development

6. Click on the **Save and manage fields** button. For now we will go with the default content type configuration for everything else.
7. Next, delete the **Body** field that is automatically added to our content type, by clicking on the **Delete** link, and then confirming by clicking on the **Delete** button on the next screen.

The screenshot shows the 'Manage fields' interface for a 'Recipe' content type. At the top, there are tabs for 'Edit', 'Manage fields' (which is selected), 'Manage form display', and 'Manage display'. Below the tabs, a success message is displayed: 'The content type Recipe has been added.' A red arrow points to this message. In the main table, there is one field entry: 'Body' (Label), 'body' (Machine Name), and 'Text (formatted, long, with summary)' (Field Type). To the right of each field entry is an 'Operations' column containing 'Edit', 'Storage settings', and a 'Delete' link. A red arrow points to the 'Delete' link in this column. The URL in the browser's address bar is 'Home > Administration > Structure > Content types > Recipe'.

8. Now, we will add some new fields to our Recipe content type. We will use the Recipe schema property names as our field names. The first property listed in the table at <http://schema.org/Recipe> is **description**. Once you click on the **Add Field** button, you will get the option **Add a new field** select box. In that you need to select **Text (formatted, long, with summary)**. On the label, you need to write **description**.

The screenshot shows the 'Field settings' page for the 'description' field. At the top, there are tabs for 'Edit' and 'Field settings' (which is selected). Below the tabs, the URL is 'Home > Administration > Structure > Content types > Recipe > Manage fields > description'. A message states: 'These settings apply to the description field everywhere it is used. These settings impact the way that data is stored in the database and cannot be changed once data has been created.' Under 'Allowed number of values', a dropdown is set to 'Limited' with a value of '1'. A red arrow points to the 'Save field settings' button, which is also circled. The URL in the browser's address bar is 'Home > Administration > Structure > Content types > Recipe > Manage fields > description'.

- 9.** Then click on **Save and continue**. On the next screen, set **Allowed number of values** limited to 1, and click on the **Save field settings** button.

The screenshot shows the 'Add field' page. At the top, there are two dropdown menus: 'Add a new field' (set to 'Text (formatted, long, with summary)') and 'Re-use an existing field' (set to '- Select an existing field -'). Below these, there is a 'Label \*' field containing 'description' with a tooltip 'Machine name: field\_description [Edit]'. At the bottom is a blue 'Save and continue' button.

- 10.** On the **description settings for Recipe** page, enter A short description of the item as **Help text**. Accept the rest of the default settings, and then click on the **Save settings** button at the bottom of the page:

The screenshot shows the 'description settings for Recipe' page. It includes a success message 'Updated field description field settings.' Below it are fields for 'Label \*' (containing 'description') and 'Help text' (containing 'A short description of the item'). There are also checkboxes for 'Required field' and 'DEFAULT VALUE'. At the bottom, the 'Save settings' button is highlighted with a red oval.

- 11.** Now we will move on to the image property. We are going to use an existing field for this property. In the **Re-use an existing field** section, select **Image: field\_image (Image)**.

The screenshot shows the 'Add field' interface in the Drupal admin. At the top, there are navigation links: Back to site, Manage, Shortcuts, and a user icon for 'admin'. Below that is a toolbar with links for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. The main area is titled 'Add field' with a star icon. It shows the path: Home > Administration > Structure > Content types > Recipe > Manage fields. There are two main options: 'Add a new field' (with a dropdown for 'Select a field type') and 'Re-use an existing field' (with a dropdown currently set to 'Image: field\_image'). A red arrow points from the text 'select an existing field' to the dropdown. Below these are fields for 'Label \*' (set to 'Image') and a 'Save and continue' button. Another red arrow points from the text 'label' to the 'Label \*' input field.

- 12.** Click on the **Save and continue** button to accept the default settings on the **FIELD SETTINGS** page.
- 13.** On the next page, click on the **Save settings** button to accept the **Image** settings for Recipe and **IMAGE FIELD SETTINGS**.
- 14.** The **DatePublished** and **Author** properties will be captured by the core Drupal node properties. We will skip the rest of the **Properties from CreativeWork** for now.
- 15.** Now, add a new number field `cookTime`, **Label** as `cookTime` field type as **Number(integer)** from the dropdown.
- 16.** Click on the **Save and continue** button to accept the default settings on the **FIELD SETTINGS** page.
- 17.** On the next page, enter `The time it takes to actually cook the dish in minutes.` as the **Help text**. Enter `minute|minutes` as the **Suffix** under the **cookTime settings for Recipe** page, and click on the **Save settings** button.

**cookTime settings for Recipe ☆**

**Edit** **Field settings**

Home > Administration > Structure > Content types > Recipe > Manage fields

✓ Updated field cookTime field settings.

**Label \***  
cookTime

**Help text**  
The time it takes to actually cook the dish in minutes.

Instructions to present to the user below this field on the editing form.  
Allowed HTML tags: <a> <b> <big> <code> <del> <em> <i> <ins> <pre> <q> <small> <span> <strong> <sub> <sup> <tt> <ol> <ul> <li> <p> <br> <img>  
This field supports tokens.

Required field

**DEFAULT VALUE**  
The default value for this field, used when creating new content.  
cookTime

**Minimum**

The minimum value that should be allowed in this field. Leave blank for no minimum.

**Maximum**

The maximum value that should be allowed in this field. Leave blank for no maximum.

**Prefix**

Define a string that should be prefixed to the value, like '\$' or '€'. Leave blank for none. Separate singular and plural values with a pipe ('pound|pounds').

**Suffix**  
minute|minutes

Define a string that should be suffixed to the value, like ' m', ' kb/s'. Leave blank for none. Separate singular and plural values with a pipe ('pound|pounds').

**Save settings** **Delete**

18. Along with **cookingMethod**, we will also skip the **nutrition**, **recipeCategory**, **recipeCuisine**, and **totalTime** properties for now. We will add these properties later in the book.
19. For the **ingredients** property, the settings will be **label: ingredients** from the **Add a new field** dropdown you can select a **Field type : Text (plain)**. Click on the **Save and continue** button. On the next screen, accept the default setting of 255 for maximum length and the **Allowed number of values** as **Unlimited**, then click on **Save field settings**.
20. On the **ingredients settings for Recipe** page, enter An ingredient used in the recipe as the **Help text**. Accept the rest of the default settings, and click on the **Save settings** button at the bottom of the page.

- 21.** For the **prepTime** property, the settings will be **label: prepTime** and from the **Add a new field** dropdown you can select a **Field type: Number(integer)**. Click on the **Save and continue** button to accept the default settings on the **FIELD SETTINGS** page. On the next page, enter the length of time it takes to prepare the recipe in minutes as the **Help text**, enter `minute|minute` as the **Suffix** under the **prepTime settings for Recipe** page, and then click on the **Save settings** button.
- 22.** Next add the **recipeInstructions** property. Click on the **Add field** button. In the next screen, select **Field type** as **Text(formatted, long)** and **Label** as **recipeInstructions**. Then click on the **Save and continue** button. On the next page, leave all the settings as default and click on the **Save** button. In the next settings page, enter `The steps to make the dish.` as the **Help text** and leave rest of the settings, and click on the **Save** button.
- 23.** For the **recipeYield** property, the settings will be **label: recipeYield** from the **Add a new field** dropdown you can select a **Field type : Text(plain)**. Click on the **Save and continue** button. On the next screen, accept the default setting of 255 for **Maximum length** and the **Allowed number of values** as **Limited to 1** then click on **Save field settings**.
- 24.** On the **recipeYield settings for Recipe** page, enter `The quantity produced by the recipe (for example, number of people served, number of servings, and so on).` as the **Help text**. Accept the rest of the default settings, and click on the **Save settings** button at the bottom of the page.
- 25.** You should now have a **Manage fields** screen for our Recipe content type that looks similar to the following screenshot:

| LABEL              | MACHINE NAME             | FIELD TYPE                           | OPERATIONS |
|--------------------|--------------------------|--------------------------------------|------------|
| cookTime           | field_cooktime           | Number (integer)                     | Edit       |
| description        | field_description        | Text (formatted, long, with summary) | Edit       |
| Image              | field_image              | Image                                | Edit       |
| ingredients        | field_ingredients        | Text (plain)                         | Edit       |
| prepTime           | field_prepTime           | Number (integer)                     | Edit       |
| recipeInstructions | field_recipeInstructions | Text (formatted, long)               | Edit       |
| recipeYield        | field_recipeyield        | Text (plain)                         | Edit       |

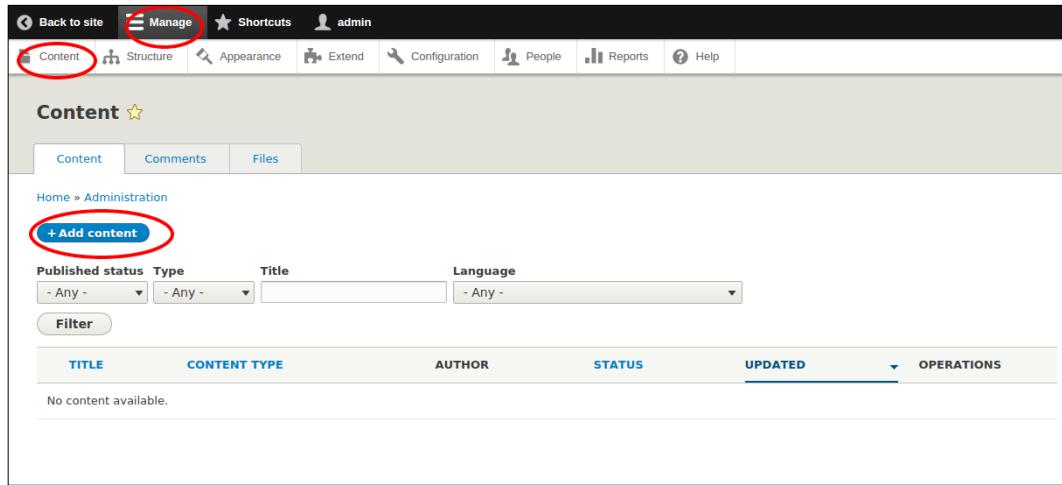
## What just happened?

You created the new Recipe content type and added new fields.

## Time for action – adding a new recipe

Now that we have created the new recipe content type and modified its fields, let's create a new recipe by clicking on the **Add content** link in the shortcut bar, and then click on the link for Recipe.

Go to the path **Manage | Content** and click on the **Add content** button, as shown in the following screenshot:



Here is my recipe for Awesome Sauce that you may use, but you are welcome to add any recipe you like:

- 1. name:** Awesome Sauce
- 2. description:** A deliciously sweet and spicy sauce that makes everything you put it on that much awesome. A little goes a long way...
- 3. ingredients:**
  - One ghost pepper (optional)
  - Two habanera peppers
  - Three Thai peppers
  - Four jalapeno peppers
  - Four garlic cloves

- Three cups of rice vinegar
- One tea spoon of fish sauce
- One cup of sugar

**4. recipeInstructions:**

- 1. Remove the stems from the peppers.
- 2. Add the peppers and garlic to a food processor, and blend until pureed.
- 3. Add vinegar, sugar, fish sauce, and puree to a small saucepan, and bring to a simmer over low heat.
- 4. Simmer sauce for 20 to 30 minutes, until the sugar has completely dissolved.
- 5. Remove the saucepan from the burner and let it stand for 10 minutes.
- 6. Your Awesome Sauce is ready to serve, or it can be refrigerated for up to three weeks.

Thai peppers and fish sauce are typically available in most Asian markets. Ghost peppers are generally considered to be the hottest pepper in the world and may be left out for those who have a little less tolerance for heat, or if you aren't able to find them.

**5. Yield:** 12 servings

**6. prepTime:** 10 minutes

**7. cookTime:** 30 minutes

Back to site   Manage   Shortcuts   admin

Content   Structure   Appearance   Extend   Configuration

**Name \***  
Awesome Sauce

**Description**

A deliciously sweet and spicy sauce that makes everything you put it on that much awesome.  
A little goes a long way...

body p

**Text format** Basic HTML   [About text formats](#)

A short description of the item

**Image**

Choose File No file chosen

One file only.  
2 MB limit.  
Allowed types: png gif jpg jpeg.

**cookTime**

30 minutes

The time it takes to actually cook the dish in minutes.

Show row weights

**INGREDIENTS**

⊕ One ghost pepper (optional)

⊕ Two habanera peppers

⊕ Three Thai peppers

When you are done, save it to view your new recipe page.

## What just happened?

We added our own recipe to the site.

## OOP concepts in Drupal

Before we dive into developing our first custom module, let's understand the basics of the modern **object-oriented programming (OOP)** approach which has been adopted in Drupal 8 to make it more familiar for developers using other pure PHP frameworks. The OOP design patterns have been used to implement a variety of Drupal concepts, such as fields, views, entities, and nodes.

Although OOP comes with a steep learning curve that includes mastering key programming techniques such as inheritance and polymorphism, it has been found to be easier to extend, refactor, and maintain in the long run when compared to the procedural programs. This lets you focus on the programming part instead of wasting time on maintenance issues. The Dependency Injection is one of the OOP design patterns that has been used extensively in Drupal 8. A basic understanding of this concept is crucial to gain access to and make use of some core APIs. Here they are:

- ◆ **Objects:** An **object** is an individual instance of the data structure defined by a class. A class is defined once and then you make an object to belong to a class. In other words, a class can be represented as a type of object. It is a blueprint from which you can create an individual object. A class is composed of three primary components: attributes, name, and operations. Here is a small PHP example:

```
<?php
class foo
{
    function call_foo()
    {
        echo "Calling foo.";
    }
}
$bar = new foo;
$bar->call_foo();
?>
```

- ◆ **Abstraction:** One of the core principles in OOP, **abstraction** refers to the representation of any type of data in which you can keep the implemented details abstracted or hidden. It allows you to write code that works seamlessly with abstract data structures such as lists, arrays, and other data types. As the code remains unchanged while you work with different data types, you can write a new data type and make it work with a program without changing it. Here is a small PHP example:

```
<?php
abstract class AbstractClass
```

```
{  
    // Force Extending class to define this method  
    abstract protected function getVal();  
    abstract protected function prefixVal($prefix);  
  
    // Common method  
    public function printOut() {  
        print $this->getVal() . "\n";  
    }  
}  
  
class ConcreteClass1 extends AbstractClass  
{  
    protected function getVal() {  
        return "ConcreteClass1";  
    }  
  
    public function prefixVal($prefix) {  
        return "{$prefix}ConcreteClass1";  
    }  
}  
  
$class1 = new ConcreteClass1;  
$class1->printOut();  
echo $class1->prefixVal('FOO_') . "\n";  
?>
```

- ◆ **Encapsulation:** Also synonymous for information hiding, encapsulation is basically the amalgamation of all the resources that are needed for an object to function, including the methods and the data. This process is carried out by creating classes that help you expose the public methods and properties. A class is seen as a capsule or a container that encapsulates the attributes and the properties along with a set of methods in order to provide indented functionalities to other classes. Here is a small PHP example:

```
<?php  
class Application {  
    private static $_user;  
    public function User( ) {  
        if( $this->_user == null ) {  
            $this->_user = new UserData();  
        }  
        return $this->_user;  
    }
```

```
}

class UserData {
    private $_name;
    public function __construct() {
        $this->_name = "Krishna kanth";
    }
    public function GetUserName() {
        return $this->_name;
    }
}
$app = new Application();
echo $app->UserData()->GetUserName();
?>
```

- ◆ **Polymorphism:** Polymorphism derives its name from *multiple shapes* and refers to the function of requesting the same operations that can be performed by different types of things. There are three major techniques that can be used to achieve polymorphism: method overloading, method overriding, and operator overloading. Here is a small PHP example:

The content of file ShapeInterface.php:

```
<?php
// Create Shape interface with calculateArea() method.
interface Shape {
    public function calculateArea();
}
?>
```

The content of file Circle.php:

```
<?php
require ShapeInterface.php;

// Create Circle class that implements Shape interface.
class Circle implements Shape {
    private $radius;

    public function __construct($radius)
    {
        $this -> radius = $radius;
    }

    // calcArea calculates the area of circles
    public function calculateArea()
```

```
{  
    return $this -> radius * $this -> radius * pi();  
}  
}  
?  
>
```

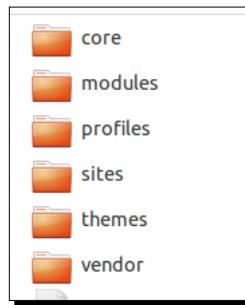
- ◆ **Inheritance:** You can create a new class from an existing class by extending it, and this process is known as **inheritance**. One of the key relationships used among objects is specialization, which is implemented using the principle of inheritance. Here is a small PHP example:

```
<?php  
// Declare the interface 'iTemplate'  
interface iTemplate  
{  
    public function setVariable($name, $var);  
    public function getHtml($template);  
}  
  
// Implement the interface  
// This will work  
class Template implements iTemplate  
{  
    private $vars = array();  
  
    public function setVariable($name, $var)  
    {  
        $this->vars[$name] = $var;  
    }  
  
    public function getHtml($template)  
    {  
        foreach($this->vars as $name => $value) {  
            $template = str_replace('{ ' . $name . ' }', $value,  
            $template);  
        }  
  
        return $template;  
    }  
}  
?  
>
```

Now we will see the preceding concepts in action while developing our new custom module.

## Time for action – developing a custom module in Drupal 8

Here we are going to create a custom module which will print Hello World in a page under the path: mypage/page. Follow the basics steps involved in the custom module development in Drupal 8. Before that, we will discuss the Drupal root directory structure.



Look at the preceding screenshot. We will see what each directory contains:

- ◆ /core: All the files provided by core that don't have an explicit reason to be in the / directory.
- ◆ /modules: The directory into which all the custom and contrib modules go. Splitting this up into the sub-directories contrib and custom can make it easier to keep track of the modules.
- ◆ /profiles: This folder contains contributed and custom profiles.
- ◆ /themes: - contributed and custom (sub)themes.
- ◆ /sites/ [domain OR default]/{modules, themes}: Site specific modules and themes can be moved into these directories to avoid them showing up on every site.
- ◆ /sites/ [domain OR default] /files: Site specific files tend to go here. This could be files uploaded by the users, such as images, but also includes the configuration, active as well as staged config. The configuration is read and written by Drupal, and should have the minimal amount of privileges required for the web server, and the only the web server, to read and modify them.
- ◆ /vendor: This folder contains all the backend libraries that Drupal Core depends on.

Now, let's begin:

1. Let's start by adding your module folder name. Unlike Drupal 7 where we used to keep the modules folder inside site/all, in Drupal 8 we have to keep our custom or contributed modules within the module available under the root directory  
`modules/contrib/`  
`modules/custom/`

 We need to use different file structures as follows in case of multisite configuration.

```
sites/ site_name_a/modules/
sites/ site_name_b/modules/
```

Let's name our module d8dev and create a folder name d8dev under the modules/custom directory.

2. Create the .info.yml file. In Drupal 7, we used to have the .info file. This has been changed to .info.yml in Drupal 8. Now instead of the .info parser, we use the Symfony YML component. And the new extension .info.yml applies to modules, themes, and profiles.

 Drupal uses some of the Symfony components for this version. By using the HTTP Kernel component, Drupal and Symfony projects became more interoperable. We will be able to easily integrate custom Symfony applications with Drupal and vice versa. You can learn more about the Symfony structure and how Drupal uses it. Follow this link <http://symfony.com/projects/drupal>.

**YAML:** Similar to PHP, YML is a simple language and there are syntaxes for simple types; for example Integers, Strings, Floats, or Booleans.

In Drupal 8, the new info.yml files are required to update Drupal core about different modules, themes, or any install profile. This also provides additional criteria to manage modules along with version compatibility.

3. Create a d8dev.info.yml file under the d8dev directory we created in Step 1:

```
name: Drupal 8 custom module d8dev
type: module
description: 'Example for Drupal 8 modules.'
package: Custom
version: 8.1
core: 8.x
```

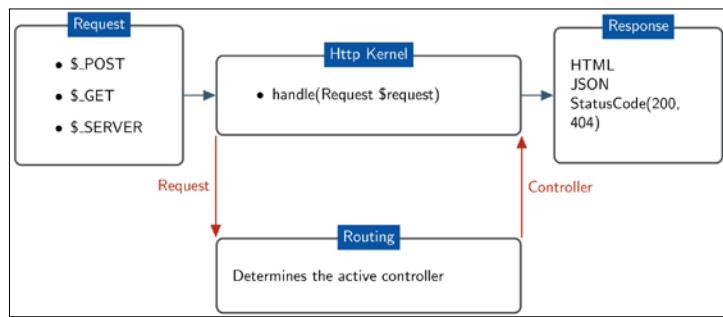
What have we added in d8dev.info.yml:

- ❑ name: As is obvious, the name field is required for our module as the identifier and will be displayed on the Module page. We should follow the convention of capitalized and Symantec names.
- ❑ description: We have added a short one-liner description to help acquaint the administrator with what this module does.

- ❑ **package:** We should use the name of the package if this is going to be part of other modules. Since we are developing a custom module, we have kept `Custom` or you can use `Other` if you are not sure.
- ❑ **type:** This is a required property to let Drupal know if it is a module, or a theme, or an installation profile.
- ❑ **version:** We should specify the version number of the module we are developing.
- ❑ **core:** We need to specify the Drupal version for which this module will be used with, `8.x`. This is a mandatory property.

#### 4. Create `.routing.yml`

- ❑ **Routing system:** A route is a path which is defined for Drupal to return some sort of content on. For example, the default front page `/node` is a route. When Drupal receives a request, it tries to match the requested path to a route it knows about. If the route is found, then the route's definition is used to return the content. Otherwise, Drupal returns a 404.
- ❑ **Routes and controllers:** Drupal's routing system works with the Symfony HTTP kernel. Reading a bit on Symfony will be helpful. Basic knowledge about the Symfony HTTP Kernel would be enough to do general route operations. The following diagram explains how the different components relate to each other:



We are going to write the path in the `.routing.yml` file. Again, here we are going to use the Symfony2 components to handle the routing. Following is our `d8dev.routing.yml` code which includes defining routes as the configuration and managing the callback in a controller, which is a method of a Controller class:

```
d8dev.my_page:  
  path: '/mypage/page'  
  defaults:
```

```

_controller:
  '\Drupal\d8dev\Controller\d8devController::myPage'
  _title: 'My first page in Drupal8'
  requirements:
    _permission: 'access content'

```

In the preceding code, the first line `d8dev.my_page` is route, a Symfony component which maps an HTTP request in Drupal to a set of configuration variables. Route is defined as a machine name as `module_name.route_name`.

Next is path, where we specify the URL path we want this route to register. Do not forget to add a leading forward slash.

We have two configurations under defaults: `_controller`, which references a method on the `d8devController` class, and `_title`, where we add the default page title, such as My first page in Drupal 8.

As part of the requirements configuration we have specified the permission, who can access the page.

You can read more on the routing file at <https://www.drupal.org/node/2092643>.

5. Create the Route Controller class. We have to create our `ModuleController.php` according to the PSR-4 naming standard which has been implemented in Drupal 8 for package-based PHP namespace auto-loading by the PHP Framework Interoperability Group.



PSR describes the specification for writing auto-loading classes from file paths. It also describes where to place the files that will be autoloaded according to the specification. Read more about PSR-4 namespaces and auto-loading at <http://www.php-fig.org/psr/psr-4/>.

Create a folder `modules/custom/d8dev/src/Controller`. Within this folder, create a file named `d8devController.php` with the following content:

```

<?php
/**
 * @file
 * @author My Name
 * Contains \Drupal\d8dev\Controller\d8devController.
 * Please include this file under your
 * d8dev(module_root_folder)/src/Controller/
 */
namespace Drupal\d8dev\Controller;
/**
 * Provides route responses for the d8dev module.
 */

```

## *Custom Module Development*

---

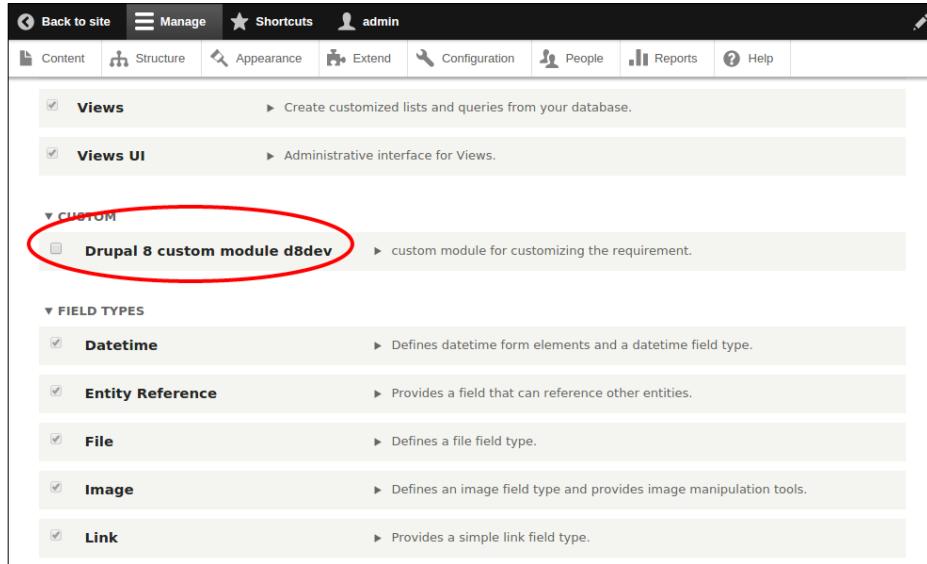
```
class d8devController {  
    /**  
     * Returns a simple page.  
     *  
     * @return array  
     * A simple renderable array.  
     */  
    public function myPage() {  
        $element = array(  
            '#type' => 'markup',  
            '#markup' => 'Hello world!',  
        );  
        return $element;  
    }  
}  
?>
```

Controller is a PHP function we add which takes the information from the HTTP request and constructs and returns an HTTP response.

The controller includes all the logic our application needs to display the content of the page. Based on the matching route, the specific controller is executed and it creates and returns a response object.

For example, if the browser requests the page having the path /mypage/page, it will execute the `d8devController::myPage()` controller and display a page that simply prints Hello world!.

6. Enable the module. Go to the path `admin/modules`.



Once you have enabled the module and opened the path `http://localhost/d8dev/mypage/page` in your favorite browser, you'll see the **Hello world!** text printed from our module.

## **What just happened?**

We completed our first custom module and displayed the **Hello World!** message on a custom page URL. Reward yourself with one hot cup of coffee before getting deeper into module development and introduction of Drupal API.

## **Time for action – developing custom field formatter**

Let's open the recipe page we created just before developing our custom module. On our recipe page which we created using our custom content type, all time fields (**cookTime** and **prepTime**) values are displayed in minutes, for example, 60 minutes and 90 minutes. It would be nice if 60 minutes was displayed as 1 hour and 90 minutes was displayed as 1 1/2 hours.

One way we can make this happen is to develop a custom module to create a custom field formatter that will display the duration related fields of **cookTime** and **prepTime** as hours instead of minutes.

Following are the basic steps involved in creating a new plugin or creating a custom field formatter:

1. Create the custom formatter class. This class defines its meta information in its annotation block, which holds the ID of the formatter, label, and field type.

Core ones are defined either by core modules that can be found inside the `Drupal\Core\Field\Plugin\Field\FieldFormatter` and namespace plugins are placed inside the `src\Plugin/` folder of our module. In the case of field formatters, this will be the `src\Plugin\Field\FieldFormatter` directory.

For creating a formatter, you need to follow these next steps:

Create `RecipeFormatter.php`. Copy this to within our module folder as `d8dev/src/Plugin/Field/FieldFormatter/RecipeFormatter.php` and add the following lines:

```
<?php
/**
 * @file
 * Contains \Drupal\d8dev\Plugin\field\formatter\RecipeFormatter.
```

```
*/  
  
namespace Drupal\d8dev\Plugin\Field\FieldFormatter;  
  
use Drupal\Core\Field\FormatterBase;  
use Drupal\Core\Field\FieldItemListInterface;
```

In Drupal 7, we used to have `hook_field_formatter_info`. Instead of that, a new annotations system has been introduced in Drupal 8. Now, we need to define the formatter using annotations.

Add the following lines to `RecipeFormatter.php` we created in step 1:

```
/**  
 * Plugin implementation of the 'recipe_time' formatter.  
 *  
 * @FieldFormatter(  
 *   id = "recipe_time",  
 *   label = @Translation("Duration"),  
 *   field_types = {  
 *     "integer",  
 *     "decimal",  
 *     "float"  
 *   }  
 * )  
 */
```

The PHP comment docblock before your formatter class with the `@FieldFormatter` annotation is really important:

 class RecipeFormatter extends FormatterBase { }

The annotation attributes are pretty self-explanatory. All we've done is define an ID, label, and which field type this formatter should be available on. The `field_types` attribute is the most important part, if you don't add `integer`, `decimal`, or `float` then this formatter will not appear on the manage display page.

The last bit of work we'll do on the formatter is add the `viewElements()` method. This method will be used to display the actual formatter and it's the only method required if the formatter class extends `FormatterBase`.

Append the following lines into the `RecipeFormatter.php` class:

```
public function viewElements(FieldItemListInterface $items) {  
  $elements = array();  
  
  foreach ($items as $delta => $item) {
```

```

// $hours = $item->value;
$hours = floor($item->value / 60);

// divide by minutes in 1 hour and get floor
$minutes = $item->value % 60;
// remainder of minutes
// get greatest common denominator of minutes to
// convert to fraction of hours
$minutes_gcd = gcd($minutes, 60);

// &frasl; is the html entity for the fraction
// separator, and
// we use the sup and sub html element to give the
// appearance of a fraction.

// $minutes_fraction = '<sup>' .
$minutes/$minutes_gcd . '</sup>&frasl;<sub>' .
60/$minutes_gcd . '</sub>';
$minutes_fraction = $minutes/$minutes_gcd . "/" .
60/$minutes_gcd ;

$markup = $hours > 0 ? $hours . ' and ' .
$minutes_fraction . ' hours' : $minutes_fraction .
' hours';
$elements[$delta] = array(
    '#theme' => 'recipe_time_display',
    '#value' => $markup,
);
}

return $elements;
}

```

What we're doing here is passing the processed value into a custom template that'll be used to display the embedded HTML code.

We also need to create a helper function to process the value.

Append the following lines to the `RecipeFormatter.php` class:

```

**
 * Simple helper function to get gcd of minutes
 */
function gcd($a, $b) {
    $b = ($a == 0) ? 0 : $b;
    return ($a % $b) ? gcd($b, abs($a - $b)) : $b;
}

```

Finally, our `RecipeFormatter.php` class looks like the following code:

```
<?php
* @file
* Contains \Drupal\d8dev\Plugin\field\formatter\RecipeFormatter.
*/


namespace Drupal\d8dev\Plugin\Field\FieldFormatter;

use Drupal\Core\Field\FormatterBase;
use Drupal\Core\Field\FieldItemListInterface;

/**
 * Plugin implementation of the 'number_decimal' formatter.
 *
 * The 'Default' formatter is different for integer fields on the
 * one hand, and
 * for decimal and float fields on the other hand, in order to be
 * able to use
 * different settings.
 *
 * @FieldFormatter(
 *   id = "recipe_time",
 *   label = @Translation("Duration"),
 *   field_types = {
 *     "integer",
 *     "decimal",
 *     "float"
 *   }
 * )
 */
class RecipeTimeFormatter extends FormatterBase {

 /**
 * {@inheritDoc}
 */
 public function viewElements(FieldItemListInterface $items) {
   $elements = array();

   foreach ($items as $delta => $item) {

     // $hours = $item->value;
     $hours = floor($item->value / 60); // divide by
     minutes in 1 hour and get floor
     $minutes = $item->value % 60; // remainder of
     minutes
   }

   return $elements;
 }

}
```

```

//get greatest common denominator of minutes to
convert to fraction of hours
$minutes_gcd = gcd($minutes, 60);

//&frasl; is the html entity for the fraction
separator, and we use the sup and sub html element
to give the appearance of a fraction.

//$minutes_fraction = '<sup>' .
$minutes/$minutes_gcd . '</sup>&frasl;<sub>' .
60/$minutes_gcd . '</sub>';
$minutes_fraction = $minutes/$minutes_gcd ."/" .
60/$minutes_gcd ;

$markup = $hours > 0 ? $hours . ' and ' .
$minutes_fraction . ' hours' : $minutes_fraction .
' hours';
$elements[$delta] = array(
    '#theme' => 'recipe_time_display',
    '#value' => $markup,
);
}

return $elements;
}
}

/**
 * Simple helper function to get gcd of minutes
 */
function gcd($a, $b) {
    $b = ($a == 0) ? 0 : $b;
    return ($a % $b) ? gcd($b, abs($a - $b)) : $b;
}

```

- 2.** Create the template. So far, now we have created the formatter. Now we need to create a custom template to complete this module development. The template will be called `recipe_time_display` and it'll accept the `value` as the single parameter. For creating the template, we will follow these next steps:

1. Open up `d8dev.module` and add the following function:

```

/**
 * Implements hook_theme().
 */
function d8dev_theme() {
    return array(

```

```
'recipe_time_display' => array(
    'variables' => array('value' => NULL),
    'template' => 'recipe-time-display',
),
);
}
```

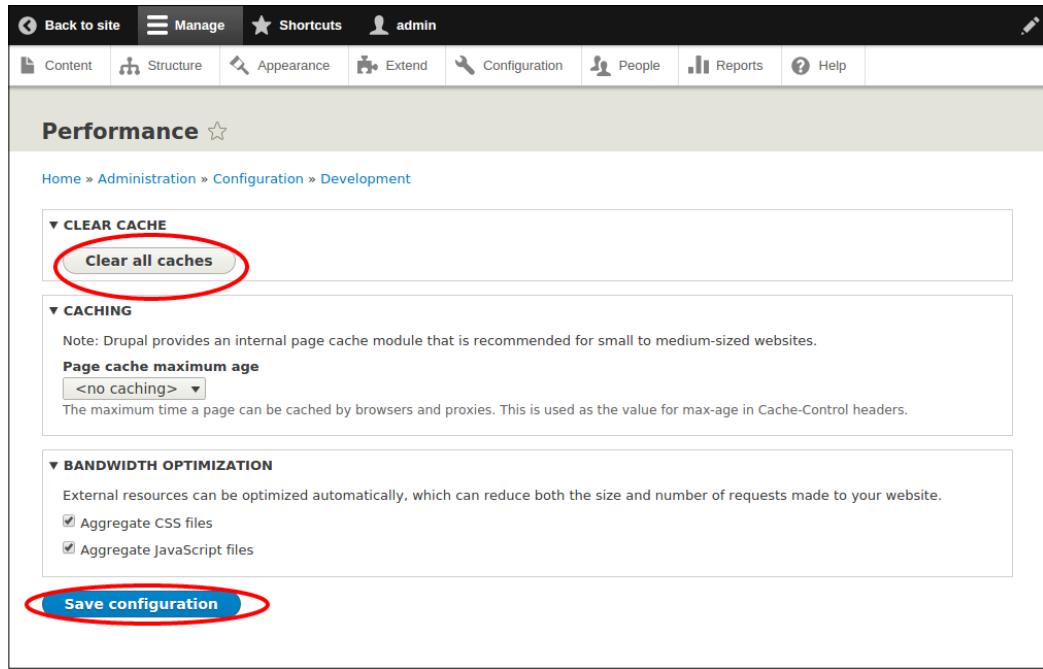
2. Create a folder in the module called templates and a file named recipe-time-display.html.twig. The path is : d8dev/templates/recipe-time-display.html.twig.

[  You can have a brief look at Twig on <http://symfony.com/doc/current/book/twig.html>. ]

Here we are just printing the value directly. If you want you can add more HTML to the value through this file. Add the following to recipe-time-display.html.twig:

```
{{ value }}
```

3. After saving all the files from the previous steps, go to your site on the browser and **Clear all caches** from the path admin/config/development/performance.



4. After clearing cache, you can go to the path `admin/structure/types/manage/recipe/display` and change the **FORMAT** fields **cookTime** and **prepTme** to the formatter from **Default** to **Duration**.

The screenshot shows the 'Manage display' interface for the 'Recipe' content type in the 'Teaser' view mode. The 'FIELD' column lists various fields: 'Links', 'description', 'Image', 'cookTime', 'ingredients', 'prepTime', 'recipeInstructions', and 'recipeYield'. The 'LABEL' column shows the field names. The 'FORMAT' column contains dropdown menus. The 'cookTime' and 'prepTime' dropdowns are highlighted with red boxes and are set to 'Duration'. The 'Links', 'description', 'Image', 'ingredients', 'recipeInstructions', and 'recipeYield' dropdowns are set to 'Visible', 'Above', 'Image', 'Plain text', and 'Default' respectively. The 'Save' button at the bottom is also highlighted with a red box.

- 5.** After that you click on the **Save** button on the **Manage display** page. Now view the Recipe content item that we created earlier, and you should see the new time format.

The screenshot shows a Drupal content page for a recipe titled "Awesome Sauce". At the top, there are four buttons: View, Edit, Delete, and Devel. Below the buttons, it says "Submitted by admin on Fri, 07/03/2015 - 11:18". The content area starts with a "description" field containing a paragraph about the sauce. Below that is a "cookTime" field with the value "1 and 1/2 hours", which is highlighted with a red box. The "ingredients" section lists various ingredients like ghost peppers, habanero peppers, Thai peppers, jalapeno peppers, garlic cloves, rice vinegar, fish sauce, and sugar. Another "prepTime" field is present with the value "1/4 hours", also highlighted with a red box. The "recipeInstructions" section contains a single step: "1. Remove the stems from the peppers."

## ***What just happened?***

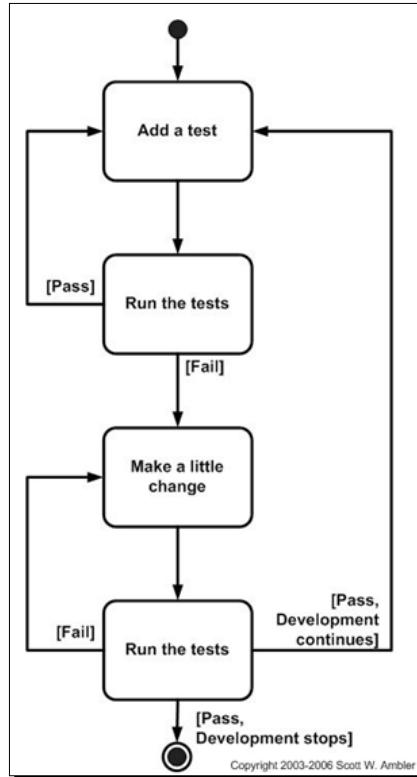
We created a custom module that allows us to format our Recipe content duration field the way we wanted it—integers converted to hours and fraction of hours.

## **Test-driven development (TDD)**

Lot of times we hear that Drupal is not a good choice for writing complex web applications. One of the chief reasons is that Drupal doesn't support TTD or automated testing. But in Drupal 8 we have the PHPUnit tests and Simpletest tests in the core.

Drupal 8 is developed keeping in mind the use of automated tests, including unit tests as well as functional tests. Unit test is at the lower level and used primarily to test the functionality of classes, whereas functional test is done at the higher level to check the web output.

Important is run both types of tests for most of the features we develop before making any changes to the existing Drupal system to make sure no existing functionality is breaking up. We also call this regression testing.



To follow TDD, we need to follow these two processes:

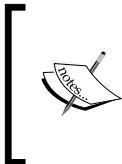
- ◆ Whenever you make a patch to fix a bug or to change an existing functionality, make sure to write a test that fails before changes have been in the code and that it passes after you make the changes in the code. The goal of this test is to help the reviewer understand the bug he/she encounters, to highlight that the code written has fixed the intended bug, and to ensure that it will not appear again whenever new changes are made to the code.
- ◆ Similarly, in the case of writing code to implement a new feature, make sure to include unit and/or functional tests in your code. This will ensure that the reviewer knows the code you have written works and that the changes will not break the new functionality.

## PHPUnit tests for Drupal classes

We are going to use an industry standard PHPUnit framework to write a test for our Drupal classes. Generally we write PHPUnit test cases to test the functionality of a class when we do not require the Drupal environment (databases, settings, and so on) and web browser.

To write a PHPUnit test, follow these next steps:

1. Define a class extending `\Drupal\Tests\UnitTestCase`. Make sure the class name ends with the word `test`.
2. Place your test class file in `d8dev/tests/src/Unit` directory.
3. Include a phpDoc comment with the description giving information about the test.
4. Start writing your test class with names starting with `test`. Each test class should include part of the functionality to be tested.



Read more about PHPUnit at <https://www.drupal.org/phpunit>.  
For full documentation on how to write PHPUnit tests for Drupal go to <http://phpunit.de> and for general information on the PHPUnit framework, object-oriented programming topics, for more on PSR-4, namespaces, and where to place classes

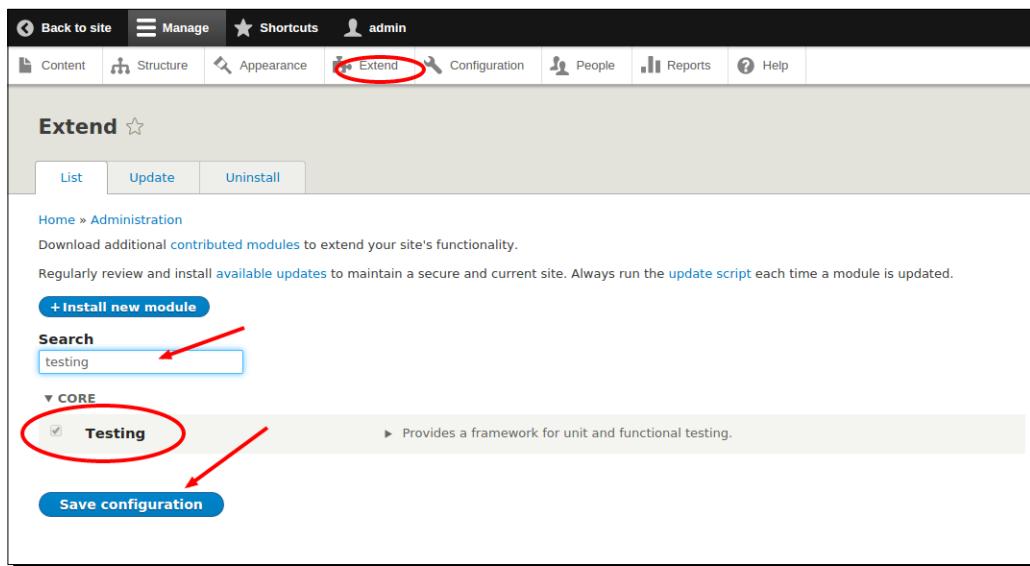


## Functional tests

We use inbuilt Simpletest as part of the Drupal core to write functional test cases. Unlike unit test cases, we use functional test to test the functionality of the different systems of Drupal when it depends on databases as well as configurations or to test the browser output.

### Time for action – writing and testing functional test from our d8dev custom module

1. Enable the module **Testing** from the path `admin/modules`:



2. Create the D8devTest class in the D8devTest.php file under d8dev/src/tests/ and add the following lines of code:

```

<?php
/**
 * Created by PhpStorm.
 * User: Neeraj
 * Date: 8/7/15
 * Time: 10:59 AM
 */
namespace Drupal\d8dev\Tests;

use Drupal\simpletest\WebTestBase;

/**
 * Tests the d8dev module functionality
 *
 * @group d8dev
 */
class D8devTest extends WebTestBase {

    /**
     * Tests that the 'mypage/page' path returns the right
     * content
     */
    public function testCustomPageExists() {

```

## Custom Module Development

```
        $this->drupalGet('mypage/page');
        $this->assertResponse(200);
    }
}
```

In the phpDoc comment block for the Test class, it is mandatory to have this block with the @group annotation.



```
/**
 * Tests the d8dev module functionality
 *
 * @group d8dev
 */
```

**Test Case:** In the D8devTest class, we check our custom page mypage/page created from our d8dev module using the function testCustomPageExists().

3. Clear the cache from the path admin/config/development/performance.
4. Open the testing module user interface from the path admin/config/development/testing or you can navigate to **Configuration | Testing**.

The screenshot shows the 'Testing' configuration page. At the top, there are tabs for 'List' and 'Settings'. Below the tabs, the URL is Home > Administration > Configuration > Development. A red arrow points to the search bar where 'd8dev' is typed. Another red arrow points to a checkbox next to the test name '\Drupal\d8dev\Tests\DrupalTest'. A blue button labeled 'Run tests' is also visible. At the bottom, there is a section for 'CLEAN TEST ENVIRONMENT' with a 'Clean environment' button.

5. Select `\Drupal\d8dev\Tests\DrupalTest` and click on the **Run tests** button. You will find the result as in the following screenshot:

The screenshot shows the Drupal 8 administration interface under the 'Testing' section. A green success message at the top states 'The test run finished in 11 sec.' Below it, an 'ACTIONS' bar includes a 'Run tests' button. The 'RESULTS' section displays a table of test outcomes. The table has columns: MESSAGE, GROUP, FILENAME, LINE, FUNCTION, and STATUS. It shows two passes (green rows), one warning (gray row with an exclamation mark), and one fail (red row with an X). The failing test is 'HTTP response expected 200, actual 404'.

| MESSAGE   | GROUP   | FILENAME      | LINE | FUNCTION  | STATUS |
|---|---------|---------------|------|---|--------|
| GET http://d8.dev/mypage/page returned 404 (1.93 KB). | Browser | D8devTest.php | 24   | Drupal\d8dev\Tests\DrupalTest->testCustomPageExists() | ✓      |
| Valid HTML found on "http://d8.dev/mypage/page"       | Browser | D8devTest.php | 24   | Drupal\d8dev\Tests\DrupalTest->testCustomPageExists() | ✓      |
| Verbose message                                       | Debug   | D8devTest.php | 24   | Drupal\d8dev\Tests\DrupalTest->testCustomPageExists() | ⚠      |
| HTTP response expected 200, actual 404                | Browser | D8devTest.php | 25   | Drupal\d8dev\Tests\DrupalTest->testCustomPageExists() | ✗      |

## What just happened?

Congratulations! You should now be familiar with TDD approach implementation in Drupal 8. We learned how to write a functional test case for our custom module functionality. We also learned how to write and run the test in Drupal 8 by testing a module user interface.

## Summary

In this chapter, we learned how to create new content type in Drupal and developed our custom module to modify one of the fields using the field formatter API. We also explored how TDD has been incorporated in the new Drupal core.

In the next chapter, we are going to learn more about newly introduced configuration management in Drupal 8 and how to use views which have been moved as core modules.



# 3

## Drupal Views and Configuration Management

*In the last chapter, we learned how to create basic custom modules as well as create new content types. Now let's explore the new configuration management introduced in Drupal 8 for better version control. We will learn how to use the Devel module to generate content for testing purposes, along with learning the basics of new and improved views.*

In these chapter, we will learn these topics:

- ◆ Using built-in Drupal 8 views
- ◆ What is new configuration management in Drupal?
- ◆ Using the configuration management module to export and import site configuration through the user interface
- ◆ Using the Devel module to autogenerate content for testing purposes

### A quick introduction to Views

Views in Drupal 7 was the most installed and most used contributed module. Using views, it is easy to create pages and blocks that list a variety of content on the site. You can create a photo gallery or an event calendar, or list out users logged in to your site. Here are a few points listed when we need views:

- ◆ We like the default front page view, but we want to sort it differently
- ◆ We want to provide an "unread forum posts" option

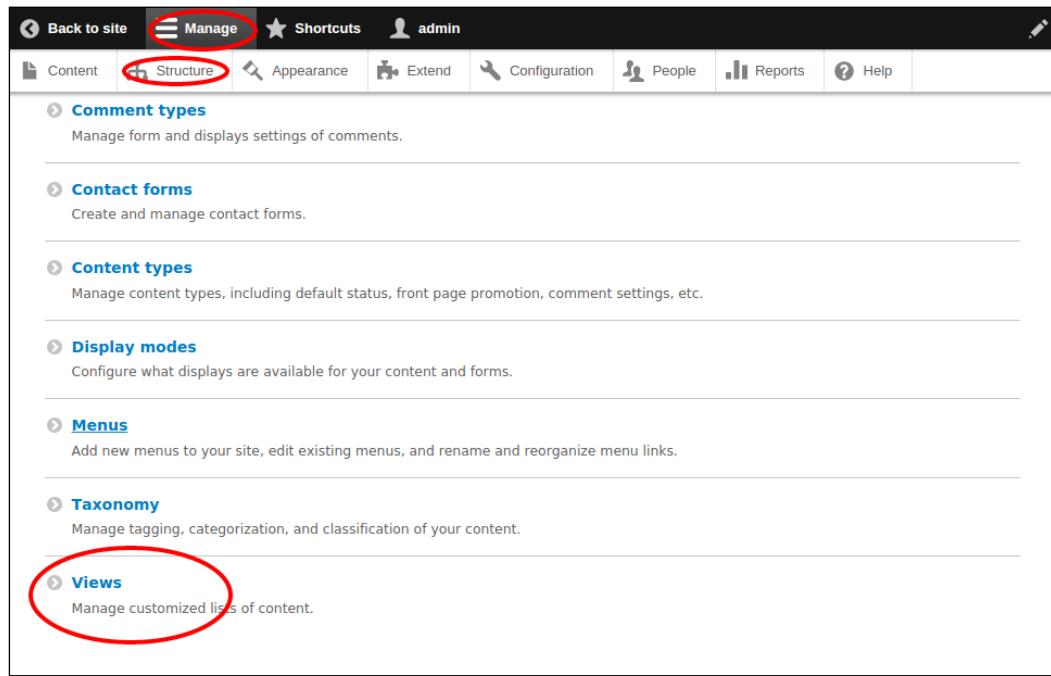
- ◆ We like the default taxonomy/term view, but we want to sort it differently, for example, alphabetically
- ◆ We want a way to display a block with the five most recent posts of a particular type

In Drupal 8, views has been moved to the core module system. Creating a view in Drupal 8 is similar to the process followed in Drupal 7. Now the Views module will get installed as part of the default core module installation.

## Time for action – creating a recipe listing block using views

We will see how easy the new Views module makes it to display a list of recipes on our site:

1. If you aren't already logged in to your site, log in as admin and navigate to **Manage | Structure** at <http://yourdrupal8site.com/admin/structure>.



2. Select the **Views** link to create a new view. It will take you to the **Views** configuration page, <http://yourdrupal8site.com/admin/structure/views>.

| VIEW NAME  | DESCRIPTION                    | TAG     | PATH  | OPERATIONS           |
|--|--------------------------------|---------|---|----------------------|
| <b>Content</b><br>Displays: Page<br>Machine name: content                    | Find and manage content.       | default | /admin/content  | <a href="#">Edit</a> |
| <b>Custom block library</b><br>Displays: Page<br>Machine name: block_content | Find and manage custom blocks. |         | /admin/structure/block/block-content                  | <a href="#">Edit</a> |
| <b>Files</b><br>Displays: Page, Page   | Find and manage files.         | default | /admin/content/files,<br>/admin/content/files/usage/% | <a href="#">Edit</a> |

Here, you will find a minor difference in style from the Drupal 7 Views configuration page. There are many default views. According to your requirement, you can either duplicate or edit and use these predefined views. Or you can create an entirely new one by clicking on the **Add new view** button.

3. Now we are going to create our own view for listing the article content type. Go to this path: <http://yourdrupal8site.com/admin/structure/views/add> or <http://yourdrupal8site.com/admin/structure/views>. Click on the **Add new view** button.
4. You will get a form similar to the one shown next. Fill in the **View name** field. In this example, I am giving the view's name as **Recipe List**. For the **Show** and **of type** fields, I have selected **Content** and **Recipe**, respectively.
5. Check the **Create a block** checkbox.

6. Click on the **Save and edit** button. Accepting all the default values for creating a block, your page should look similar to this:

The screenshot shows the 'Add new view' configuration page in Drupal. The 'View name' field is set to 'Recipe List'. The 'Create a block' checkbox is checked. The 'Save and edit' button is highlighted with a red circle.

7. It will take you to the views configuration page at [http://yourdrupal8site.com/admin/structure/views/view/recipe\\_list](http://yourdrupal8site.com/admin/structure/views/view/recipe_list). There, you have to click on the **Save** button at the bottom of the page:

The screenshot shows the 'Displays' configuration page for a 'Block' view. The 'Display name' is set to 'Block'. Under 'TITLE', the 'Title' is 'Recipe List'. In 'FORMAT', the 'Format' is 'Unformatted list'. Under 'FIELDS', there is a single field 'Content: Title'. In 'FILTER CRITERIA', there are two filters: 'Content: Publishing status (Yes)' and 'Content: Type (= Recipe)'. Under 'SORT CRITERIA', there is one sort criterion: 'Content: Authored on (desc)'. On the right side, under 'BLOCK SETTINGS', the 'Block name' is 'None' and the 'Block category' is 'Lists (Views)'. Other settings include 'Allow settings: Items per page', 'Access: Permission | View published content', 'HEADER', 'FOOTER', 'NO RESULTS BEHAVIOR', and 'PAGER' (set to 'Use pager: Display a specified number of items | 5 items'). At the bottom, the 'Save' button is highlighted with a red circle.

Now, we need to configure our d8dev site so that our Recipe List views-based block shows up on the front page.

**8. Click on the **Structure** link in the **Admin** toolbar and select **Block layout**:**

The screenshot shows the 'Structure' page. The 'Structure' link in the top navigation bar is circled in red. Below it, under the heading 'Block layout', there is a sub-link 'Block layout' which is also circled in red. The page lists several other configuration options: 'Comment types', 'Contact forms', 'Content types', 'Display modes', and 'Menus'.

9. On the next page, click on the **+Recipe List** on the right-hand side of the page under **LISTS (VIEWS)**:

The screenshot shows the 'Block layout' configuration page in the Drupal admin interface. The main area displays various regions (Header, Primary menu, Secondary menu, Help, Featured top, Breadcrumb, Content, Sidebar first, Sidebar second) with their respective block settings. To the right, a sidebar titled 'Place blocks' lists several categories: DEVEL, FORMS, HELP, and a expanded section for LISTS (VIEWS). Within the LISTS (VIEWS) section, the '+Recipe List' item is highlighted with a red box and a red arrow pointing to it from the top right.

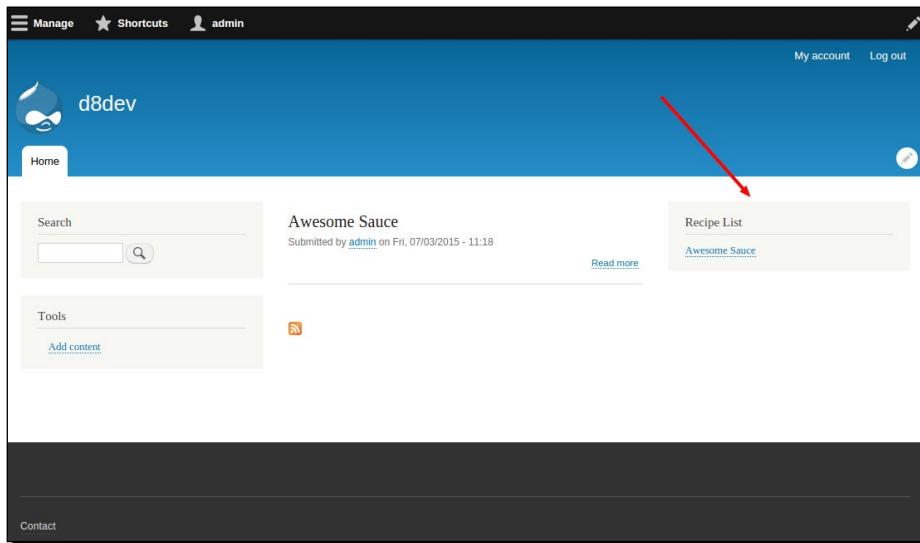
10. On the next page, leave the **Block title** field blank because that way the title will default to the title we added previously in the view creation wizard.

11. Under the **Region** settings, select **Sidebar second** from the dropdown:

The screenshot shows the 'Configure block' page for a 'Recipe List' block. At the top, there are tabs for Back to site, Manage, Shortcuts, and admin. Below the tabs, the breadcrumb navigation shows Home > Administration > Structure > Block layout > Configure block. The main content area has a title 'Configure block' with a star icon. Underneath, it says 'Block description: Recipe List'. There is a checked checkbox for 'Display title'. A section titled 'CACHE SETTINGS' is collapsed. The 'Items per block' setting is set to '5 (default setting)'. An unchecked checkbox for 'Override title' is present. A 'Visibility' section contains two tabs: 'Content types' (selected) and 'Pages'. Under 'Content types', 'Not restricted' is selected. Under 'Pages', 'Not restricted' is selected. Under 'Roles', 'Not restricted' is selected. A 'Machine-readable name' field is set to 'views\_block\_recipe\_list\_block\_1'. A note below it says 'A unique name for this block instance. Must be alpha-numeric and underscore separated.' A 'Region' dropdown is set to 'Sidebar second' and is highlighted with a red box. A note below it says 'Select the region where this block should be displayed.' At the bottom right, a blue 'Save block' button is shown with a red arrow pointing to it.

## What just happened?

You have now created a views-based block of recipes and configured it so that it will only be displayed on the front page. Now, when you visit the front page of our d8dev site, you will have a nice **Recipe List** block on the right-hand side of the page:



## Configuration management in Drupal 8

As developers, we have always have faced the situation of how to do incremental development or work around staging and deployment issues in Drupal projects. An even bigger challenge has been, "What to classify as content and what as configuration?" Though we have configuration as a separate menu to identify this problem, what about views, taxonomy, or theme configuration?

It's not that we do not have a solution in Drupal 7. To address this issue, the Features & Strongarm module was introduced during the Drupal 6 days, and it has made great improvements in the last couple of years. This was a savior in most of the cases. Still, we had to recreate a few configurations manually, making it error-prone.

But documenting the configuration required for the feature export itself has always been challenging when there is user-generated content being added periodically, and we have multiple teams working on the same project, sometimes in different time zones. We cannot export the complete site configuration from one installation to another.

Besides, there are many contributed modules that store specific configurations in their own ways, making it more difficult to consolidate feature exports in an incremental manner.

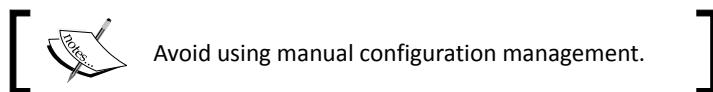
To improve the development process, a new configuration management initiative was started in Drupal 8. Now, because of this, we have an easier way to manage configuration in Drupal sites. We have the entire configuration saved in files instead of database tables. This makes it easier to follow version control systems in the true way. Now we can store history and roll back changes if required.

Let's start by understanding what has been defined as content and what is configuration in a Drupal project:

- ◆ **Content:** Everything that we display on the site's frontend is content, for example, articles, basic pages, blog posts, images, video, and audio
- ◆ **Sessions:** Sessions store information about specific users when logged in while they are interacting with the Drupal site
- ◆ **State:** This holds the information about the site in different scenarios that are temporary in nature, for example, the state when site permission is rebuilt or before a cron job is run
- ◆ **Configuration:** This is all of the information that is not content and is specific to your website, for example, the site name, content types used, fields, vocabularies, image styles, text formats, menu, menu links, permission system, and even the views you have created

It is best to use a version control system, for example, `git` or `svn`, to make best use of configuration management, especially when you are working with a distributed team.

Besides adding configuration in files, you can also add configuration manually. But this is best avoided as you will have to remember every step to be recreated during migration from staging to deployment. You cannot even use any version control system.

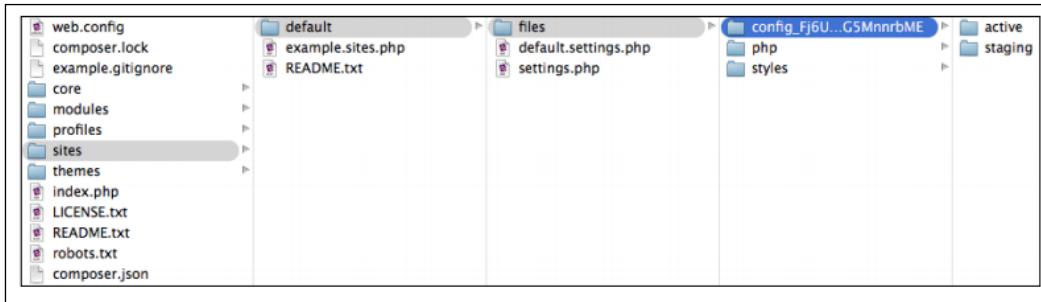


## Using the Configuration Management interface

In Drupal 8 we have a Configuration Manager module, which provides an easy interface to export and import site configurations and share between different environments of the project, such as development, staging, and deployment. You can easily verify the changes from your production environment using this module.

This module assumes your different environment; for example, dev, staging, testing, and production are the same site. Each site is identified using a **Universally Unique Identifier (UUID)**. The module will allow importing of configuration between different sites only when the UUID matches the target site.

By default, Drupal 8 saves all site configurations in the database. When installing the new Drupal site, it creates a new folder in `sites/default/files` named `config_HASH`. Here, HASH is a randomly generated long string of numbers and letters. Refer to this screenshot:



This is where the configuration of different states of the site is saved.

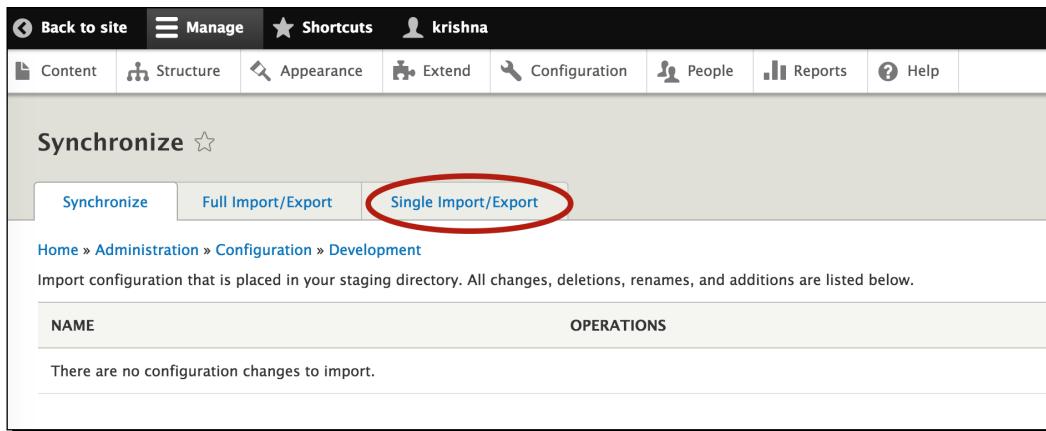
Let us use the module UI to export the configuration we created in the previous chapter in the following section.

## Time for action – importing, exporting, and synchronizing configurations

The goal of the configuration system in Drupal 8 is to make taking a copy of the site configuration easy to set up a development site where you make changes. Then, importing those changes to the other site is also made simple.

To export Recipe content type, follow these steps:

1. In *Chapter 2, Custom Module Development*, we created the Recipe content type. We will export this content type using configuration management. Browse to the module page at <http://yourdrupal8site.com/admin/config/development/configuration>.



The screenshot shows the Drupal Configuration Management interface. At the top, there's a navigation bar with links for Back to site, Manage, Shortcuts, and a user profile for krishna. Below the navigation is a horizontal menu with tabs: Synchronize, Full Import/Export, and Single Import/Export. The Single Import/Export tab is highlighted with a red oval. The main content area has a title 'Synchronize' with a star icon. Below it, there are three tabs: Synchronize, Full Import/Export, and Single Import/Export. Underneath these tabs, the breadcrumb navigation shows Home > Administration > Configuration > Development. A message states 'Import configuration that is placed in your staging directory. All changes, deletions, renames, and additions are listed below.' A table header with columns 'NAME' and 'OPERATIONS' is shown, followed by a message 'There are no configuration changes to import.'

2. Click on the **Single import/Export** tab and then **Export**.

On the configuration management module UI, the first tab is **Synchronize**. When you are using this feature for the first time, you will see: **There are no configuration changes to import**. This implies that your dev site is using the configurations from the database itself.

You will see that there are no configuration changes, which means that your site is using the configuration files from the database and that no changes were made to the site.

**3.** Select the **Configuration type** as **Content type** and **Configuration name** as **Recipe**.

You will see YAML-formatted configurations in the text area box. Save these YAML configurations in a separate file, `node.type.recipe.yml`. This file contains all the configurations related to our content type.

The screenshot shows the 'Single export' page under 'Configuration > Development > Synchronize'. The 'Export' tab is active. The 'Configuration type' dropdown is set to 'Content type' and is circled in red. The 'Configuration name' dropdown is set to 'Recipe' and is also circled in red, with a red arrow pointing to it from the left. The text area below displays the YAML configuration for the 'Recipe' content type, which includes fields like 'uuid', 'langcode', 'status', 'dependencies', 'module', 'third\_party\_settings', 'menu\_ui', 'available\_menus', 'parent', 'name', 'type', 'description', 'help', 'new\_revision', 'preview\_mode', and 'display\_submitted'. At the bottom, it says 'Filename: node.type.recipe.yml'.

```

uuid: c5e5d8a8-5f21-4618-9800-e38ab4584b4f
langcode: en
status: true
dependencies:
  module:
    - menu_ui
third_party_settings:
  menu_ui:
    available_menus:
      - main
    parent: 'main'
  name: Recipe
  type: recipe
  description: ''
  help: ''
  new_revision: false
  preview_mode: 1
  display_submitted: true

```

Filename: `node.type.recipe.yml`

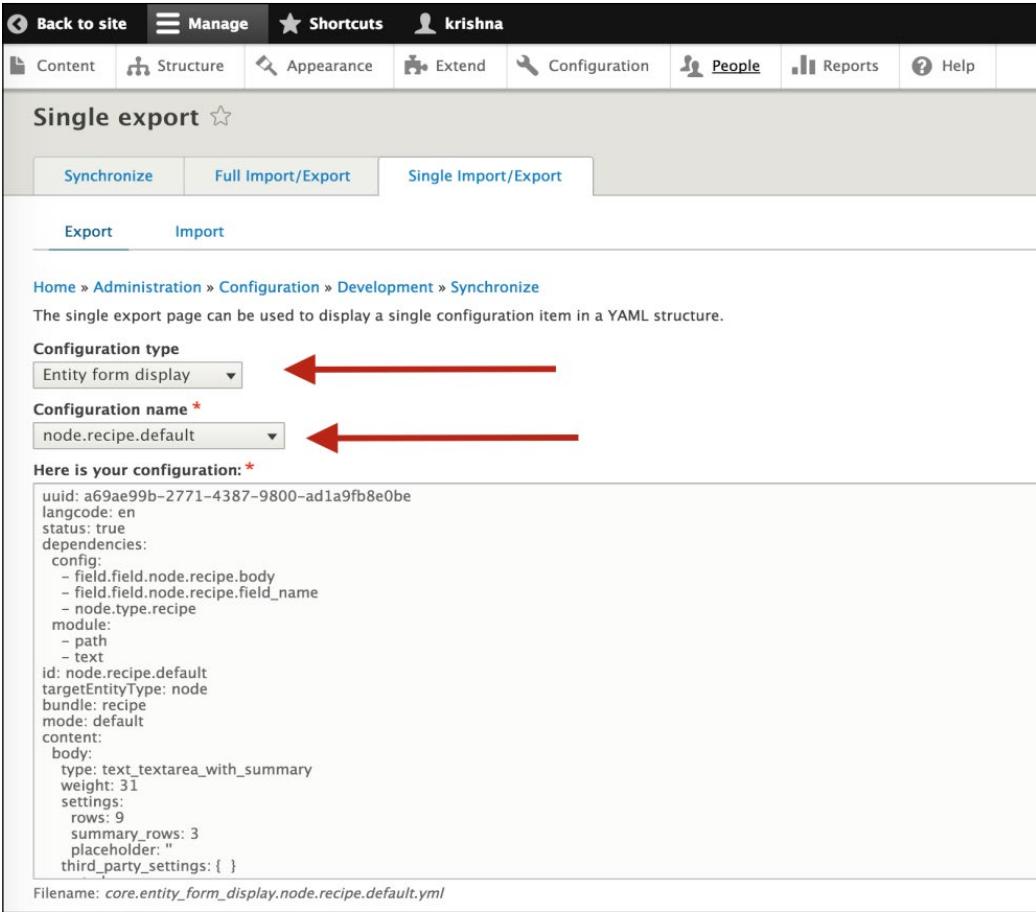
However, `node.type.recipe.yml` doesn't include any fields used in the same. We need to export them too.

4. To export, select **Field** from the dropdown on the same page.

The screenshot shows the 'Single export' page in the Drupal administration interface. The top navigation bar includes links for Back to site, Manage, Shortcuts, krishna, Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. Below the navigation is a breadcrumb trail: Home > Administration > Configuration > Development > Synchronize. The main content area is titled 'Single export' with a star icon. It has tabs for Synchronize, Full Import/Export, and Single Import/Export, with 'Single Import/Export' selected. Below the tabs are 'Export' and 'Import' buttons, with 'Export' selected. A red arrow points to the 'Configuration type' dropdown menu, which is set to 'Field'. Another red arrow points to the 'Configuration name' dropdown menu, which is set to 'Name'. The text 'Here is your configuration:' is followed by a code block containing YAML configuration for a field named 'Name'. At the bottom of the code block is the text 'Filename: field.field.node.recipe.field\_name.yml'.

```
uuid: 2d21a4a4-730e-441e-a800-adfeca28908d
langcode: en
status: true
dependencies:
  config:
    - field.storage.node.field_name
    - node.type.recipe
id: node.recipe.field_name
field_name: field_name
entity_type: node
bundle: recipe
label: Name
description: ""
required: false
translatable: false
default_value:
  -
    value: ""
default_value_callback: ""
settings: { }
field_type: string_long
```

5. In the same way, export the form display of the Recipe content type. Select the **Configuration type as Entity form display** and **Configuration name as node.recipe.default**. Save the YAML configuration in `core.entity_form_display.node.recipe.default.yml`.



The screenshot shows the 'Single export' page in the Drupal configuration interface. The 'Configuration type' dropdown is set to 'Entity form display' and the 'Configuration name' dropdown is set to 'node.recipe.default'. Red arrows point from the text descriptions below to these two dropdown fields. The page displays a YAML configuration snippet for the entity form display.

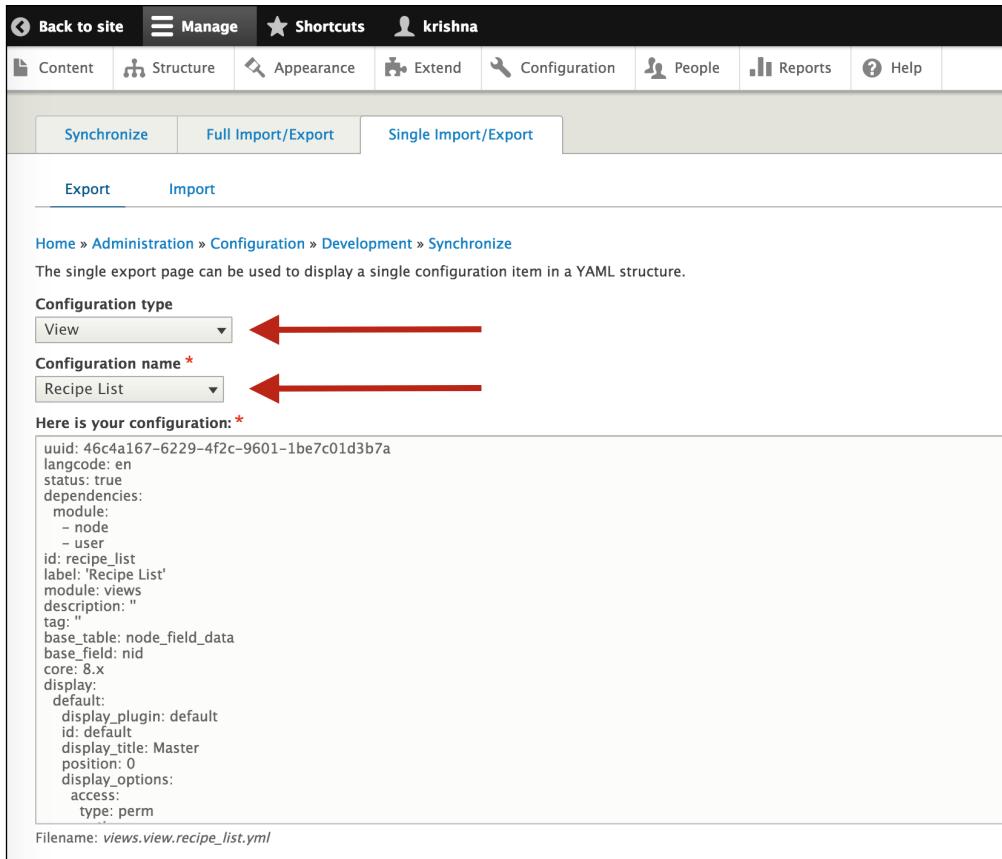
```

uuid: a69ae99b-2771-4387-9800-ad1a9fb8e0be
langcode: en
status: true
dependencies:
  config:
    - field.field.node.recipe.body
    - field.field.node.recipe.field_name
    - node.type.recipe
  module:
    - path
    - text
id: node.recipe.default
targetEntityType: node
bundle: recipe
mode: default
content:
  body:
    type: text_textarea_with_summary
    weight: 31
    settings:
      rows: 9
      summary_rows: 3
      placeholder: ''
      third_party_settings: { }
  
```

Filename: core.entity\_form\_display.node.recipe.default.yml

6. Export the default view teaser of the Recipe content type. Select the **Configuration type** as **Entity view display** and **Configuration name** as **node.recipe.teaser**. Save the YAML configuration in **core.entity\_view\_display.node.recipe.default.yml**.

7. Export the view Recipe List. Select the **Configuration type** as **View** and **Configuration name** as **Recipe List**. Save the YAML configuration in `views.view.recipe_list.yml`.



The screenshot shows the Drupal Configuration Synchronization page. At the top, there are tabs for 'Synchronize', 'Full Import/Export', and 'Single Import/Export'. The 'Single Import/Export' tab is selected. Below it, there are two tabs: 'Export' (which is selected) and 'Import'. The main content area shows the configuration details for a 'View' named 'Recipe List'. A red arrow points to the 'Configuration type' dropdown menu, which is set to 'View'. Another red arrow points to the 'Configuration name' dropdown menu, which is set to 'Recipe List'. The configuration details are displayed below these fields. The configuration details are as follows:

```
uuid: 46c4a167-6229-4f2c-9601-1be7c01d3b7a
langcode: en
status: true
dependencies:
  module:
    - node
    - user
id: recipe_list
label: 'Recipe List'
module: views
description: ""
tag: ""
base_table: node_field_data
base_field: nid
core: 8.x
display:
  default:
    display_plugin: default
    id: default
    display_title: Master
    position: 0
    display_options:
      access:
        type: perm
```

Filename: `views.view.recipe_list.yml`

### Importing configurations in other dev sites

Install another Drupal site on your local machine which we will be using as target site to import all the features. Now import the content type Recipe from the configuration management:

1. Go to **Administration | Configuration | Development | Synchronize | Import** at `http://yourstagingsite.com/admin/config/development/configuration/single/import`. Select the **Configuration type** as **Content type**, paste the YAML from `node.type.recipe.yml`, and click on **Import**.

The screenshot shows the 'Single Import/Export' page in the Drupal administration interface. The 'Import' tab is highlighted with a red circle and a red arrow points to the 'Content type' dropdown menu. The configuration YAML code is pasted into the text area below.

```

uuid: c5e5d8a8-5f21-4618-9800-e38ab4584b4f
langcode: en
status: true
dependencies:
  module:
    - menu_ui
third_party_settings:
  menu_ui:
    available_menus:
      - main
    parent: 'main:'
name: Recipe
type: recipe
description: ''
help: ''
new_revision: false
preview_mode: 1
display_submitted: true
  
```

**Configuration type \***

**Paste your configuration here \***

**ADVANCED**

**Import**

You will get a confirmation page asking you to create a new content type.

The screenshot shows a confirmation dialog box. It asks, "Are you sure you want to create a new *recipe* content type? ☆". Below the dialog are the 'Synchronize', 'Full Import/Export', and 'Single Import/Export' tabs. The 'Import' tab is selected. The confirmation message also states: "This action cannot be undone."

**Synchronize**   **Full Import/Export**   **Single Import/Export**

**Import**

Are you sure you want to create a new *recipe* content type? ☆

This action cannot be undone.

**Confirm**   **Cancel**

On successful import, you should see the following screen:

The screenshot shows the Drupal configuration synchronization interface. At the top, there are tabs for 'Synchronize', 'Full Import/Export', and 'Single Import/Export'. Below these, there are 'Export' and 'Import' buttons. The URL in the browser is 'Home > Administration > Configuration > Development > Synchronize'. A green success message box contains the text: '✓ The node\_type Recipe was imported.' Below the message, there is a note: 'The single import page can be used to import a single configuration item by pasting a YAML structure into the text field.' There is a dropdown menu labeled 'Configuration type \*' with the option '- Select -' and a text input field labeled 'Paste your configuration here \*'.

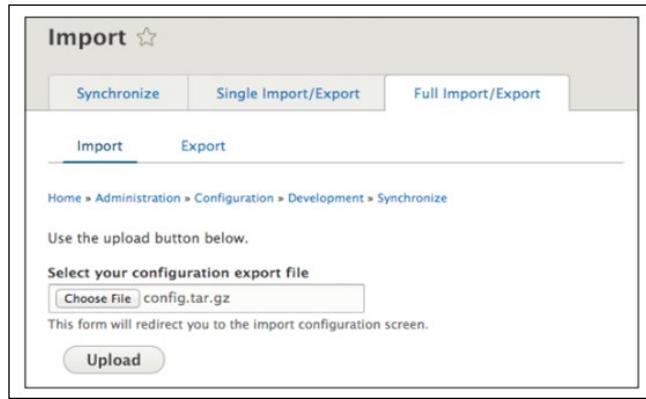
2. In a similar way, import fields of the Recipe content type. Select the **Configuration type** as **Field**, paste the YAML configurations, and click on **Import**.
3. Import the view modes and form modes using the file we saved after exporting from our initial dev site.

In the preceding steps, we saw how **Single Import/Export** works. Let's explore the other option of **Full Import/Export**:

1. Browse to the module configuration page again: <http://yourdrupal8site.com/admin/config/development/configuration>. This time we will focus on the third tab, **Full Import/Export**. As earlier, there are two options, **Import** and **Export**.

The screenshot shows the 'Export' tab selected on the configuration synchronization interface. The URL is 'Home > Administration > Configuration > Development > Synchronize'. Below the tabs, there are 'Import' and 'Export' buttons. A note says 'Use the export button below to download your site configuration.' A large 'Export' button is visible at the bottom.

2. Export your site configuration by clicking on the option. This will create a config.tar.gz file and prompt you to download the same on your machine.
3. The next step would be to create another Drupal 8 site, if you have not created one already, where we will be importing the config.tar.gz file.
4. Browse to <http://yourstagingsite.com/admin/config/development/configuration/full/import>. Select config.tar.gz and upload it as shown in this screenshot:



5. Now that we have imported our site configuration to the new site, let's check out the first tab, **Synchronize**, again. Visit <http://yourstagingsite.com/admin/config/development/configuration>. Now it should display something like this:

| NAME            | OPERATIONS                       |
|-----------------|----------------------------------|
| update.settings | <a href="#">View differences</a> |

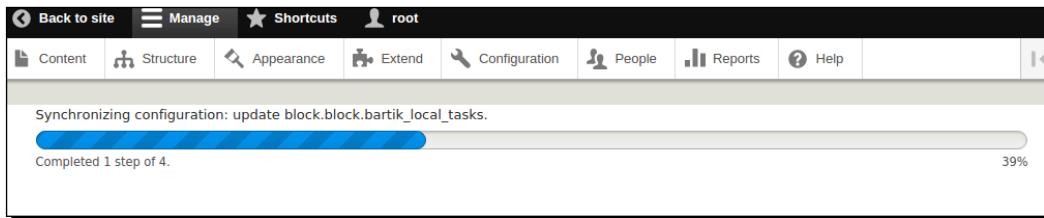
  

| NAME                             | OPERATIONS                       |
|----------------------------------|----------------------------------|
| block.block.bartik_branding      | <a href="#">View differences</a> |
| block.block.bartik_help          | <a href="#">View differences</a> |
| block.block.bartik_local_actions | <a href="#">View differences</a> |

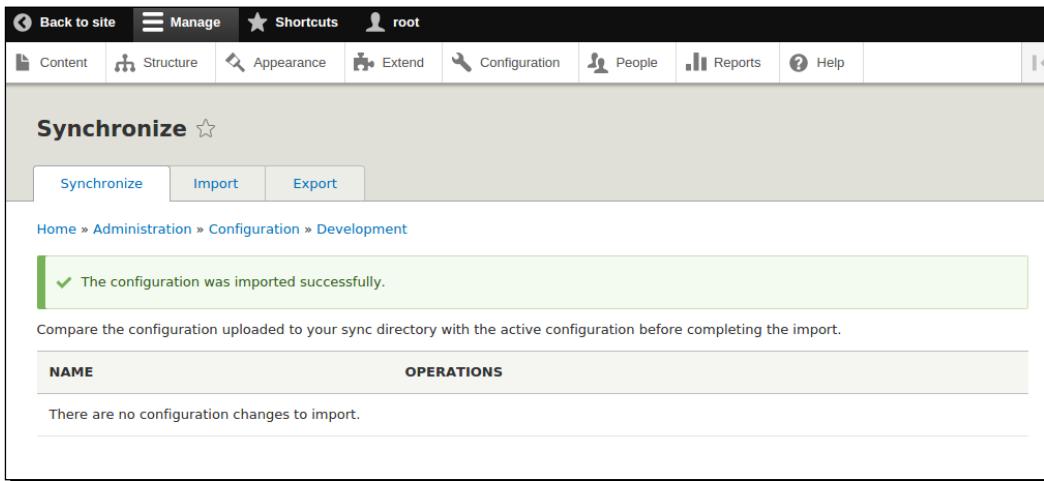
This page is basically displaying all the configuration changes. It helps us to verify that everything was imported as expected.

You can click on **View differences**, which will open a popup listing out the various configurations imported/changed. After making sure that everything is as expected, close the pop-up window and select **Import all**.

This will take a couple of minutes depending on the volume of tasks.



6. Once the import is complete, you should be back on the configuration page, in the first tab and with a success message.



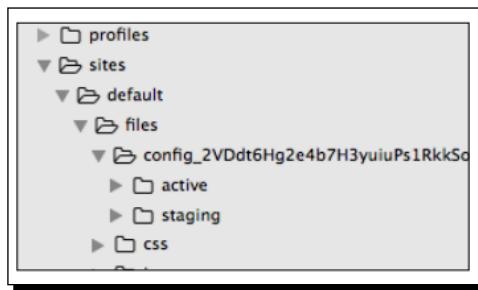
Now, just verify that the new configuration has been implemented.

## **What just happened?**

You have imported the Recipe content type from your dev site to a new instance of the Drupal site using an easy-to-use configuration management module UI, without using database import or the features module.

## **Working of Configuration Management in Drupal 8**

As developers, let's explore how configuration management works in Drupal 8. As mentioned earlier in the chapter, by default, Drupal keeps all configurations in the database. While installing, it creates a folder called `config_HASH` within `/sites/default/files`, where `HASH` is a randomly generated long string consisting of numbers and letters. Randomly generated `HASH` ensures additional protection for the website.



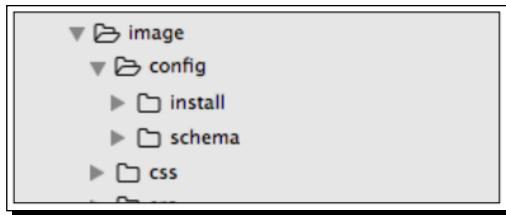
Within the `Config` folder, there are two additional folders: `active` and `staging`. Both are empty by default and contain only `.htaccess` and `README.txt`.

Generally, Drupal 8 uses the database to store the active configuration, unless we change it. If we change the default behavior, the `active` folder will contain the configuration of the default Drupal installation.

Another directory, `staging`, is used to store the configuration of the Drupal site imported from your dev environment.

By default, Drupal and the associated module store the configuration under active storage during installation. These configurations are sets of files that maintain the configuration required to run your Drupal site. These configurations are stored in YAML format.

If you explore the Drupal installation folder on your machine, you will find that there is a config folder under each module and profile. This folder contains two or more folders, install and schema.



Both the folders contain the set of .yml files that hold the respective configuration for the module or profiles.

During the site installation process, values stored in these files are copied to the active configuration of your Drupal site. In case we are using default Drupal configuration storage, the values will be copied to config tables, and if we have selected file-based storage, these configuration values will be copied to respective directories in active.

## Changing the active configuration storage

Even though it is not advisable to change the default active configuration storage from database to files, you can make this change in your settings.php file.



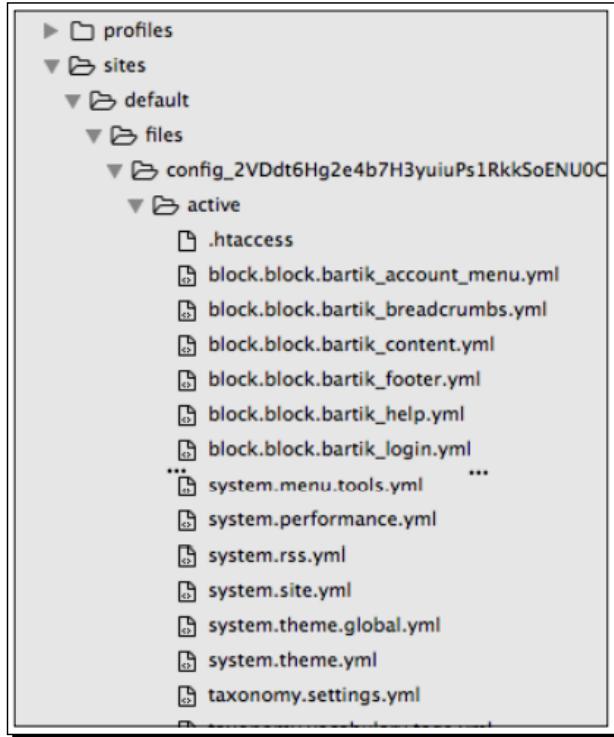
You can make this change only before installing your Drupal site. You will not be able to change the configuration storage after completion of the installation process.

Open settings.php in your editor and search for active configuration settings. Now you just have to uncomment the line \$settings['bootstrap\_config\_storage'] to enable the file-based system for active configuration. In addition to this, you also have to copy default.services.yml to services.yml and enable to file-based configuration storage:

```
services:  
  # Override configuration storage.  
  config.storage:  
    class: Drupal\Core\Config\CachedStorage  
    arguments: ['@config.storage.active', '@cache.config']  
  config.storage.active:  
    # Use file storage for active configuration.  
    alias: config.storage.file
```

This will enable Drupal to change the default configuration storage setting from the database to a file-based system and use the `config.storage.file` as active storage.

After changing the configuration storage setting, install your Drupal 8 site by following the same standard steps. Now let's have a look at our `active` folder under `sites/default/files`.



Now the active directory contains the site configuration of the entire site. These files have been copied here during the installation process. Now, whenever you make the change to your site configuration, changes will be made in these files too. You can use a version control system to track changes in your site actively.

But changing the storage system will not change the way we export/import the site configuration.



Learn more about configuration management at <https://www.drupal.org/documentation/administer/config>.

## Introducing the Devel module

So, in the previous section, we saw how easy it was to create a custom Views-based block for displaying a list of the recipes on the front page. One thing that you will notice right away is that there is only one recipe showing up, because so far we have only created one.

You may also recall that when we created the Recipe List block with Views, we left the setting for items per page at the default value of 5. Now, it would be nice to be able to test that setting without needing to manually create four more recipe items. Enter the Devel module at <http://drupal.org/project/devel>.

The Devel module includes a number of submodules that make Drupal development easier; and the one we are interested in to help us out with content creation for development purposes is the `devel_generate` module.

## Installing the Devel module

This should be an easy step. Just download the module and enable the **Extend** page.

The screenshot shows the 'Extend' page in the Drupal administration interface. At the top, there are tabs for 'Content', 'Structure', 'Appearance', 'Extend' (which is highlighted with a red circle), 'Configuration', 'People', 'Reports', and 'Help'. Below the tabs, there are two buttons: 'List' and 'Uninstall'. The main content area has a heading 'Extend' with a star icon. It displays the message: 'Home » Administration' and 'Download additional contributed modules to extend your site's functionality.' A search bar contains the text 'devel'. Below the search bar, there is a text input field with the placeholder 'Enter a part of the module name or description'. Under the heading '▼ DEVELOPMENT', there are four listed modules, each with a brief description and a red circle around its name: 'Devel' (description: 'Various blocks, pages, and functions for developers.'), 'Devel generate' (description: 'Generate dummy users, nodes, menus, taxonomy terms...'), 'Devel Kint' (description: 'Wrapper for Kint debugging tool'), and 'Devel Node Access' (description: 'Developer blocks and page illustrating relevant node\_access records.'). At the bottom of the page is a blue 'Install' button.

Devel is installed and ready to do some magic.

## Time for action – generating dummy content using the devel\_generate module

Now, with the devel\_generate module enabled, we are going to generate some Recipe content so that we can test the number of items in our Recipe List block:

1. First, click on the **Configuration** link in the admin toolbar, and then click on the **Generate content** link under the **Development** section.
2. On the **Generate content** page, uncheck all the **Content type** checkboxes except the one for **Recipe**.

| CONTENT TYPE                               | COMMENTS          |
|--|-------------------|
| <input type="checkbox"/> Article           | • Comments: Open  |
| <input type="checkbox"/> Basic page        | No comment fields |
| <input checked="" type="checkbox"/> Recipe | No comment fields |

Delete all content in these content types before generating new content.

How many nodes would you like to generate? \*

50

How far back in time should the nodes be dated?

1 week ago ▾

Node creation dates will be distributed randomly from the current time, back to the selected time.

3. Stick with the default values for all the rest of the settings, and click on the **Generate** button at the bottom of the page.

4. Now, navigate to the home page and you will see the **Recipe List** block fully populated.



### ***What just happened?***

Although this is a very simple example of using the `devel_generate` module, being able to generate content can be a big time saver when testing custom code that requires multiple content items. We just used the `devel_generate` module to generate some dummy content, based on our custom Recipe content type, and now our **Recipe List** block on the home page is fully populated with five recipes.

## **Summary**

In this chapter, we learned to create view blocks using the views module, which is a part of the Drupal 8 core now. We also explored newly introduced configuration management for better version control of our Drupal site. Now we can set up the development, staging, and production environments separately and add new features to the site incrementally. We explored how to use the Devel module to generate a lot of content for testing purposes.

In the next chapter, we are going to learn about HTML5, which is one of the five major initiatives outlined by Dries for Drupal 8.

# 4

## Introduction to the Field Types API and Developing the Custom Field Module

*In the last chapter, we learned how to use views and about configuration management. Now let's understand more about the Field Types API and how to use it to develop custom field modules.*

In this chapter, we will learn these topics:

- ◆ Creating field types, widgets, and formatters using field APIs
- ◆ Developing custom field modules

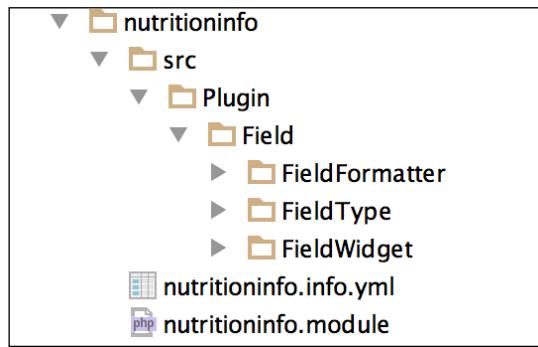
### Introducing the NutritionInformation module

In the previous chapter, one of the `http://schema.org/Recipe` properties that we did not include with our Drupal Recipe content type is the `NutritionInformation` property. The reason for this is that the `NutritionInformation` property is itself an `itemType` from `http://schema.org`, and as such is made up of a number of its own individual properties. In order to add `NutritionInformation` to our custom Recipe content type, we will need to create a custom Drupal compound field module that is based on the specification at `http://schema.org/NutritionInformation`.

## Time for action – developing a custom module for a compound NutritionInformation field

Rather than adding the code to create this compound field to our existing module, we are going to create a new module, as it is possible that it is something that could be useful to the Drupal community as a whole, and we may want to eventually contribute it to Drupal (<https://www.drupal.org/>). We need to create a compound field that consists of several different form elements (a select number and several text fields). We can now have a basic compound field in three basic steps: first is defining the field (info and schema), second is defining the field form (widget), and then comes defining the output (formatter). All of these steps are explained here:

1. In PhpStorm, create a new folder named `nutritioninfo` in the `/modules/custom` directory.
2. Create the `nutritioninfo.module` and `nutritioninfo.info.yml` files with the same name as the folder—`nutritioninfo`. Create the templates `src/Plugin/Field` folders and you should have a folder structure that looks similar to the following screenshot:



3. Now, open the `nutritioninfo.info` file and add the following configuration:

```
name: Nutrition Information
type: module
description: 'Defines a nutrition information field type based on
the Microdata spec at http://schema.org/NutritionInformation'
package: Fields
core: '8.x'
```

4. In Drupal 7, the `hook_field_schema` hook allows us to define a database schema for storing our custom field information. But in Drupal 8, field types changed to a plugin system (<https://www.drupal.org/node/2064123>).

- 5.** Next, we are going to use the following code from the telephone core

module (/core/modules/telephone/src/Plugin/Field/FieldType/  
TelephoneItem.php). We are only copying this file and changing to the  
NutritioninfoItem.php file. In the next step, we create the /modules/  
nutritioninfo/src/Plugin/Field/FieldType/NutritioninfoItem.php  
file and core TelephoneItem.php file, which looks like this:

```
<?php

/**
 * @file
 * Contains \Drupal\telephone\Plugin\Field\FieldType\
TelephoneItem.
 */

namespace Drupal\telephone\Plugin\Field\FieldType;

use Drupal\Core\Field\FieldDefinitionInterface;
use Drupal\Core\Field\FieldItemBase;
use Drupal\Core\Field\FieldStorageDefinitionInterface;
use Drupal\Core\TypedData\DataDefinition;

/**
 * Plugin implementation of the 'telephone' field type.
 *
 * @FieldType(
 *   id = "telephone",
 *   label = @Translation("Telephone number"),
 *   description = @Translation("This field stores a telephone
number in the database."),
 *   category = @Translation("Number"),
 *   default_widget = "telephone_default",
 *   default_formatter = "basic_string"
 * )
 */
class TelephoneItem extends FieldItemBase {

 /**
 * {@inheritDoc}
 */
 public static function schema(FieldStorageDefinitionInterface
$field_definition) {
 return array(
 'columns' => array(
 'value' => array(

```

```
        'type' => 'varchar',
        'length' => 256,
    ),
),
);
}

/**
 * {@inheritDoc}
 */
public static function
propertyDefinitions(FieldStorageDefinitionInterface
$field_definition) {
    $properties['value'] = DataDefinition::create('string')
        ->setLabel(t('Telephone number'))
        ->setRequired(TRUE);

    return $properties;
}

/**
 * {@inheritDoc}
 */
public function isEmpty() {
    $value = $this->get('value')->getValue();
    return $value === NULL || $value === '';
}

/**
 * {@inheritDoc}
 */
public function getConstraints() {
    $constraint_manager = \Drupal::typedDataManager()->getValidationConstraintManager();
    $constraints = parent::getConstraints();

    $max_length = 256;
    $constraints[] = $constraint_manager->create('ComplexData',
array(
    'value' => array(
        'Length' => array(
            'max' => $max_length,
            'maxMessage' => t('%name: the telephone number may
not be longer than @max characters.', array('%name'
=> $this->getFieldDefinition()->getLabel(), '@max' =>
```

```

        $max_length) ,
    )
),
));
}

return $constraints;
}

/**
 * {@inheritDoc}
 */
public static function
generateSampleValue(FieldDefinitionInterface
$field_definition) {
    $values['value'] = rand(pow(10, 8), pow(10, 9)-1);
    return $values;
}
}

```

As we discussed in step 4, Drupal 8 introduced a plugin system. Most plugins in Drupal 8 use annotations to register themselves and describe their metadata. Here, `@FieldType` is the annotation it is registering to the `telephone` field type. The `schema` function holds columns and its properties such as `type` and `length`.

The `PropertyDefinitions` function is for setting column property values such as `label` and `description`. The `isEmpty` function determines whether the list contains any nonempty items.

6. Drupal 8 implemented the PSR-4 standard for package-based PHP namespace auto-loading by the PHP Framework. Each module has a namespace that corresponds to its module name (`namespace Drupal\telephone\Plugin\Field\FieldType`). In our module, it should be `namespace Drupal\nutritioninfo\Plugin\Field\FieldType`. And the module's namespace is mapped to the `./src/` folder in the module directory. Classes and interfaces with a backslash (\) inside their fully qualified name (for example, use `Drupal\Core\Field\FieldItemBase;`) must not use their fully qualified name inside the code. If the namespace differs from the namespace of the current file, put a `use` statement at the top of the file. Here is an example:

```

namespace Drupal\nutritioninfo\Plugin\Field\FieldType;
use Drupal\Core\Field\FieldItemBase;
use Drupal\Core\Field\FieldStorageDefinitionInterface;
use Drupal\Core\TypedData\DataDefinition;

```

- 7.** Now, we are going to create the `/modules/nutritioninfo/src/Plugin/Field/FieldType/NutritioninfoItem.php` file. We need to add the `NutritioninfoItem` class. The `NutritioninfoItem.php` code looks as follows without any methods added:

```
<?php
/**
 * @file
 * Contains \Drupal\custom_field\Plugin\Field\FieldType\NutritioninfoItem.
 */

namespace Drupal\nutritioninfo\Plugin\Field\FieldType;

use Drupal\Core\Field\FieldItemBase;
use Drupal\Core\Field\FieldStorageDefinitionInterface;
use Drupal\Core\TypedData\DataDefinition;

/**
 * Plugin implementation of the 'nutritioninfo' field type.
 *
 * @FieldType(
 *   id = "nutritioninfo",
 *   label = @Translation("Nutrition Information"),
 *   description = @Translation("A field type used for storing nutrition information as defined by the Microdata spec at http://schema.org/ NutritionInformation."),
 *   default_widget = "nutritioninfo_standard",
 *   default_formatter = "nutritioninfo_default"
 * )
 */

class NutritioninfoItem extends FieldItemBase
{}
```

Here, the `@FieldType` annotation registers the `nutritioninfo` field type.

- 8.** Next, we need to add schema columns in the `schema` function. In `TelephoneItem`, we have only one column value. But in `NutritioninfoItem`, there are 12 columns need to be added. So that `schema` function looks as follows. Add these functions inside the `NutritioninfoItem` class:

```
 /**
 * {@inheritDoc}
```

```
/*
public static function
schema(FieldStorageDefinitionInterface $field)
{
    return array(
        'columns' => array(
            'calories' => array(
                'type' => 'varchar',
                'length' => 256,
                'not null' => FALSE,
            ),
            'carbohydrate_content' => array(
                'type' => 'text',
                'length' => 256,
                'not null' => FALSE,
            ),
            'cholesterol_content' => array(
                'type' => 'varchar',
                'length' => 256,
                'not null' => FALSE,
            ),
            'fat_content' => array(
                'type' => 'varchar',
                'length' => 256,
                'not null' => FALSE,
            ),
            'fiber_content' => array(
                'type' => 'varchar',
                'length' => 256,
                'not null' => FALSE,
            ),
            'protein_content' => array(
                'type' => 'varchar',
                'length' => 256,
                'not null' => FALSE,
            ),
            'saturated_fat_content' => array(
                'type' => 'varchar',
                'length' => 256,
                'not null' => FALSE,
            ),
            'serving_size' => array(
                'type' => 'varchar',
                'length' => 256,
            )
        )
    );
}
```

```
        'not null' => FALSE,
    ) ,
    'sodium_content' => array(
        'type' => 'varchar',
        'length' => 256,
        'not null' => FALSE,
    ) ,
    'sugar_content' => array(
        'type' => 'varchar',
        'length' => 256,
        'not null' => FALSE,
    ) ,
    'trans_fat_content' => array(
        'type' => 'varchar',
        'length' => 256,
        'not null' => FALSE,
    ) ,
    'unsaturated_fat_content' => array(
        'type' => 'varchar',
        'length' => 256,
        'not null' => FALSE,
    ) ,
),
);
}
```

- 9.** Next, we need to add the `isEmpty()` function. This determines whether the list contains any nonempty items. Add these functions inside the `NutritioninfoItem` class:

```
/***
 * {@inheritDoc}
 */
public function isEmpty()
{
    $calories = $this->get('calories')->getValue();
    $carbohydrate_content = $this
        ->get('carbohydrate_content')->getValue();
    $cholesterol_content = $this
        ->get('cholesterol_content')->getValue();
    $fat_content = $this->get('fat_content')->getValue();
    $fiber_content = $this->get('fiber_content')
        ->getValue();
    $protein_content = $this->get('protein_content')
```

```

->getValue();
$saturated_fat_content = $this
->get('saturated_fat_content')->getValue();
$serving_size = $this->get('serving_size')->getValue();
$sodium_content = $this->get('sodium_content')
->getValue();
$sugar_content = $this->get('sugar_content')
->getValue();
$trans_fat_content = $this->get('trans_fat_content')
->getValue();
$unsaturated_fat_content = $this
->get('unsaturated_fat_content')->getValue();

//the nutrition field is empty if all of its properties
are empty
return empty($calories)
&& empty($carbohydrate_content)
&& empty($cholesterol_content)
&& empty($fat_content)
&& empty($fiber_content)
&& empty($protein_content)
&& empty($saturated_fat_content)
&& empty($serving_size)
&& empty($sodium_content)
&& empty($sugar_content)
&& empty($trans_fat_content)
&& empty($unsaturated_fat_content);
}

```

- 10.** The last method in this class is `propertyDefinitions()`, which is used to set column property values such as `label` and `description`. Add these functions inside the `NutritioninfoItem` class:

```

/**
 * {@inheritDoc}
 */
public static function
propertyDefinitions(FieldStorageDefinitionInterface
$field_definition)
{
    $properties['calories'] =
DataDefinition::create('string')
->setLabel(t('Calories'))
->setDescription(t('The number of calories.'));
}

```

```
$properties['carbohydrate_content'] =
DataDefinition::create('string')
->setLabel(t('Carbohydrate Content'))
->setDescription(t('The number of grams of
carbohydrates.'));

$properties['cholesterol_content'] =
DataDefinition::create('string')
->setLabel(t('Cholesterol Content'))
->setDescription(t('The number of milligrams of
cholesterol.'));

$properties['fat_content'] =
DataDefinition::create('string')
->setLabel(t('Fat Content'))
->setDescription(t('The number of grams of fat.'));

$properties['fiber_content'] =
DataDefinition::create('string')
->setLabel(t('Fiber Content'))
->setDescription(t('The number of grams of
fiber.'));

$properties['protein_content'] =
DataDefinition::create('string')
->setLabel(t('Protein Content'))
->setDescription(t('The number of grams of
protein.'));

$properties['saturated_fat_content'] =
DataDefinition::create('string')
->setLabel(t('Saturated Fat Content'))
->setDescription(t('The number of grams of
saturated fat.'));

$properties['serving_size'] =
DataDefinition::create('string')
->setLabel(t('Serving Size'))
->setDescription(t('The serving size, in terms of
the number of volume or mass.'));

$properties['sodium_content'] =
DataDefinition::create('string')
->setLabel(t('Sodium Content'))
```

```

        ->setDescription(t('The number of milligrams of
            sodium.'));

    $properties['sugar_content'] =
    DataDefinition::create('string')
        ->setLabel(t('Sugar Content'))
        ->setDescription(t('The number of grams of
            sugar.'));

    $properties['trans_fat_content'] =
    DataDefinition::create('string')
        ->setLabel(t('Trans Fat Content'))
        ->setDescription(t('The number of grams of trans
            fat.'));

    $properties['unsaturated_fat_content'] =
    DataDefinition::create('string')
        ->setLabel(t('Unsaturated Fat Content'))
        ->setDescription(t('The number of grams of
            unsaturated fat.'));

    return $properties;
}

```

- 11.** Finally, our NutritioninfoItem.php code looks as shown here: <https://raw.githubusercontent.com/valuebound/nutritioninfo/master/src/Plugin/Field/FieldType/NutritioninfoItem.php>.
- 12.** Next, we need to tell Drupal how to handle our compound field on the node edit form. In Drupal 7, we have `hook_field_widget_info` for making Drupal aware of our custom widget, and then `hook_field_widget_form` for actually adding the form components to the node form. In Drupal 8, field widgets have become plugins. Now, we are going to add the `FieldWidget/NutritioninfoDefaultWidget.php` file in the `/modules/nutritioninfo/src/Plugin/Field/FieldWidget` folder. And the file looks like this without any methods:

```

<?php
/**
 * @file
 * Contains \Drupal\custom_field\Plugin\Field\FieldWidget\
NutritioninfoDefaultWidget.
 */
namespace Drupal\nutritioninfo\Plugin\Field\FieldWidget;

```

```
use Drupal\Core\Field\FieldItemListInterface;
use Drupal\Core\Field\WidgetBase;
use Drupal\Core\Form\FormStateInterface;

/**
 * Plugin implementation of the 'nutritioninfo_default' widget.
 *
 * @FieldWidget(
 *   id = "nutritioninfo_default",
 *   label = @Translation("Nutrition Field Widget"),
 *   field_types = {
 *     "nutritioninfo"
 *   }
 */
class NutritioninfoDefaultWidget extends WidgetBase {

}
```

The `FieldItemListInterface` interface is for fields, being lists of field items. This interface must be implemented by every entity field, whereas contained field items must implement `FieldItemInterface`. Moreover, `FormStateInterface` provides an interface for an object containing the current state of a form. It will be used to store information related to the processed data in the form.

Here, the `@FieldWidget` annotation registers the `nutritioninfo_default` widget.

- 13.** Next, we need to add the `formElement()` function. In this function, form elements will be added, which are displayed in the node edit/add form for the nutrition field. We are taking one form element to understand:

```
$element['calories'] = array(
    '#title' => t('Calories'),
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]->calories)
        ? $items[$delta]->calories : NULL,
);
```

This is the `calories` `textfield`. It will display as a `textfield` while editing/adding the form. The `formElement()` function's code looks like this:

```
/**
 * {@inheritDoc}
 */
public function formElement(FieldItemListInterface $items,
    $delta, array $element, array &$amp;form, FormStateInterface
```

```

$form_state) {
$element['calories'] = array(
    '#title' => t('Calories'),
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]->calories)
        ? $items[$delta]->calories : NULL,
);
$element['carbohydrate_content'] = array(
    '#title' => t('Carbohydrate Content'),
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]
->carbohydrate_content) ? $items[$delta]
->carbohydrate_content : NULL,
);
$element['cholesterol_content'] = array(
    '#title' => t('Cholesterol Content'),
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]
->cholesterol_content) ? $items[$delta]
->cholesterol_content : NULL,
);
$element['fat_content'] = array(
    '#title' => t('Fat Content'),
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]
->fat_content) ? $items[$delta]->fat_content :
NULL,
);
$element['fiber_content'] = array(
    '#title' => t('Fiber Content'),
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]
->fiber_content) ? $items[$delta]->fiber_content :
NULL,
);
$element['protein_content'] = array(
    '#title' => t('Protein Content'),
    '#type' => 'textfield',
    '#default_value' => isset($items[$delta]
->protein_content) ? $items[$delta]
->protein_content : NULL,
);

```

```
$element['saturated_fat_content'] = array(
  '#title' => t('Saturated Fat Content'),
  '#type' => 'textfield',
  '#default_value' => isset($items[$delta]
->saturated_fat_content) ? $items[$delta]
->saturated_fat_content : NULL,
);

$element['serving_size'] = array(
  '#title' => t('Serving Size'),
  '#type' => 'textfield',
  '#default_value' => isset($items[$delta]
->serving_size) ? $items[$delta]->serving_size :
NULL,
);

$element['sodium_content'] = array(
  '#title' => t('sodium Content'),
  '#type' => 'textfield',
  '#default_value' => isset($items[$delta]
->sodium_content) ? $items[$delta]->sodium_content
: NULL,
);

$element['sugar_content'] = array(
  '#title' => t('Sugar Content'),
  '#type' => 'textfield',
  '#default_value' => isset($items[$delta]
->sugar_content) ? $items[$delta]->sugar_content :
NULL,
);

$element['trans_fat_content'] = array(
  '#title' => t('Trans Fat Content'),
  '#type' => 'textfield',
  '#default_value' => isset($items[$delta]
->trans_fat_content) ? $items[$delta]
->trans_fat_content : NULL,
);

$element['unsaturated_fat_content'] = array(
  '#title' => t('Unsaturated Fat Content'),
  '#type' => 'textfield',
```

```

        '#default_value' => isset($items[$delta]
      ->unsaturated_fat_content) ? $items[$delta]
      ->unsaturated_fat_content : NULL,
    ) ;

    return $element;
}

```

- 14.** The full file, `NutritioninfoDefaultWidget.php`, looks as shown here: <https://raw.githubusercontent.com/valuebound/nutritioninfo/master/src/Plugin/Field/FieldWidget/NutritioninfoDefaultWidget.php>.
- 15.** Now, we need to format our compound field when displaying a content item. In Drupal 7, we used the `hook_field_formatter_info` and `hook_field_formatter_view` hooks. Now in Drupal 8, these hooks are used as plugins. We are going to add the `NutritioninfoDefaultFormatter.php` file in the `/modules/nutritioninfo/src/Plugin/Field/FieldFormatter` folder. And the code looks like this without any methods:

```

<?php
/**
 * @file
 * Contains \Drupal\custom_field\Plugin\field\formatter\NutritioninfoDefaultFormatter.
 */

namespace Drupal\nutritioninfo\Plugin\Field\FieldFormatter;

use Drupal\Core\Field\FormatterBase;
use Drupal\Core\Field\FieldItemListInterface;

/**
 * Plugin implementation of the 'nutritioninfo_default' formatter.
 *
 * @FieldFormatter(
 *   id = "nutritioninfo_default",
 *   label = @Translation("Nutritioninfo Formatter"),
 *   field_types = {
 *     "nutritioninfo"
 *   }
 * )
 */
class NutritioninfoDefaultFormatter extends FormatterBase
{
}

```

Here, the `@FieldFormatter` annotation is registering the `nutritioninfo` formatter.

- 16.** Next, we need to add the `viewElements()` function. This builds a renderable array for table theme markup. The `viewElements()` function's code looks as follows:

```
/**  
 * {@inheritDoc}  
 */  
public function viewElements(FieldItemListInterface $items)  
  
{  
    $rows = array();  
    foreach ($items as $delta => $item) {  
        $rows[] = array(  
            'data' => array(  
                $item->calories,  
                $item->carbohydrate_content,  
                $item->cholesterol_content,  
                $item->fat_content,  
                $item->fiber_content,  
                $item->protein_content,  
                $item->saturated_fat_content,  
                $item->serving_size,  
                $item->sodium_content,  
                $item->sugar_content,  
                $item->trans_fat_content,  
                $item->unsaturated_fat_content  
            )  
        );  
    }  
  
    $headers = array(  
        t('Calories'),  
        t('Carbohydrate Content'),  
        t('Cholesterol Content'),  
        t('Fat Content'),  
        t('Fiber Content'),  
        t('Protein Content'),  
        t('Saturated Fat Content'),  
        t('Serving Size'),  
        t('Sodium Content'),  
        t('Sugar Content'),  
    );  
}
```

```

        t('Trans Fat Content'),
        t('Unsaturated Fat Content'),
    ) ;

$table = array(
    '#type' => 'table',
    '#header' => $headers,
    '#rows' => $rows,
    '#empty' => t('No calories information available'),
    '#attributes' => array('id' => 'nutrition-info'),
)
;

return $elements = array('#markup' => drupal_
render($table));

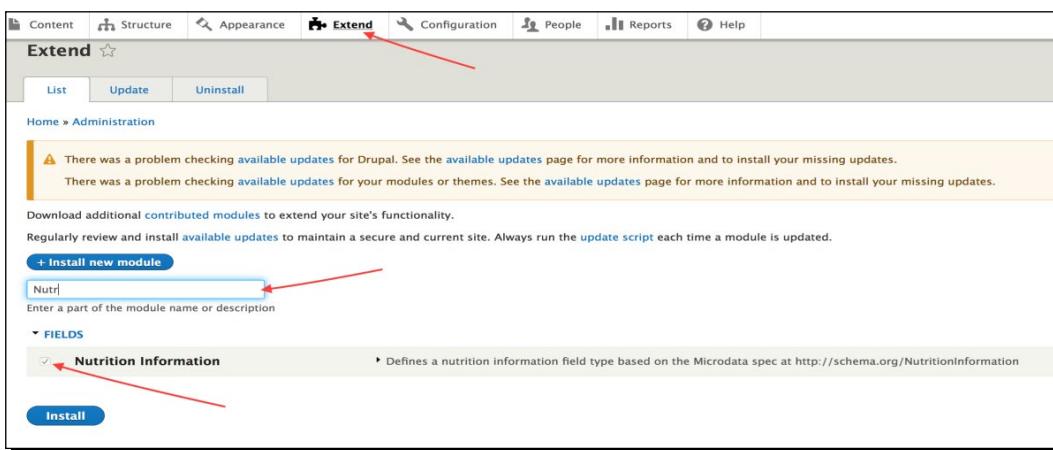
}

```

- 17.** The code of the full NutritioninfoDefaultFormatter.php file is shown here:

<https://github.com/valuebound/nutritioninfo/blob/master/src/Plugin/Field/FieldFormatter/NutritioninfoDefaultFormatter.php>.

- 18.** All right, now it's time to enable our new module. Open up your d8dev Drupal site in your favorite browser, and click on the **Modules** link in the **Admin** toolbar. Search for Nutrition and check the checkbox next to your new **Nutrition Information** field module:



- 19.** Finally, scroll to the bottom of the page and click on the **Install** button.

## **What just happened?**

That was some serious development. We created a fairly complex custom module, and now we have a field that offers a more complete Recipe content type. We used the Field Types API and created a new custom field, nutrition. In this process, we learned how to create FieldType, Field, and Field formatter, which are required for the nutrition field. We also learned about Drupal namespaces and how to use them in custom modules.

## **Time for action – updating the Recipe content type to use the NutritionInformation field**

Now, let's put our new module to use and add our new compound field to our Recipe content type, using our new custom nutritioninfo field for the NutritionInformation property of the `http://schema.org/Recipe` definition:

- 1.** Go to the **Manage Fields** configuration page for the Recipe content type: `http://localhost/d7dev/admin/structure/types/manage/recipe/fields`.
- 2.** Now, add a new field with the following settings—**Label**: nutrition, **Name**: `field_nutrition_information`, and **Type**: Nutrition Information. Click on the **Save and continue** button:

The screenshot shows the 'Add a new field' dialog. It has two main sections: 'Add a new field' and 'Re-use an existing field'. Under 'Add a new field', there is a dropdown menu showing 'Nutrition Information' and a 'Label \*' input field containing 'nutrition'. Below these are 'Machine name' and '[Edit]' links. A blue 'Save and continue' button is at the bottom. Under 'Re-use an existing field', there is a dropdown menu showing '- Select an existing field -'.

- 3.** There is nothing to set for the field settings, so just click on the **Save field settings** button.
- 4.** Next, click on the **Find content** link in the **Shortcuts** toolbar, and click on the **Edit** link for the first Recipe content item in the list.
- 5.** Towards the bottom of the node edit form, you will see inputs for our new compound field.

## **What just happened?**

We completed the development of a new module. It will create a nutrition field programmatically, which will display the data in a specific format as required.

## **Summary**

In this chapter, we developed a custom compound field module based on the microdata specification at <http://schema.org/NutritionInformation>, allowing us to enhance our Recipe content type. However, at this point, our Nutrition Information field module is still a bit rough around the edges.

In the next chapter, we will introduce some more code examples, and clean up some of those rough edges.



# 5

## Theming in Drupal 8

*In the last chapter, we learned about the Field Types API and about developing a custom field module. Now let's explore the frontend by creating a custom theme and learning about the new features of Drupal 8's theming layer.*

In this chapter, we will learn the following:

- ◆ What a theme is and how to create a sub-theme
- ◆ What is new about themes in Drupal 8
- ◆ What the terms mobile first and responsive mean and why D8 base themes are built this way
- ◆ How to install Drush
- ◆ How to add assets to your theme
- ◆ An introduction to the Twig templating language
- ◆ Understanding the benefits of contributed modules and when to make use of them

### What is a theme?

A **theme** adjusts, or overrides, the default appearance of your Drupal site. It does this by giving us the ability to change the markup (with templating), functionality (with JavaScript), and appearance (with CSS). Theming is the final layer of development before the work is viewed by the user. The icing on the cake!

By default, Drupal 8 comes shipped with a number of themes in core. Let's have a look at them.

In PhpStorm, go to the root directory of your project and then navigate to core | themes. In this directory, you'll find five themes: Bartik, Classy, Seven, Stable, and Stark. What are these and why are they here?

Let's discuss these core themes we have available by default, as they each have a distinct use:

- ◆ Bartik has been the default theme for both Drupal 7 and Drupal 8. It's well documented, well maintained, and contains some useful theme functions.
- ◆ Seven has been the default administration theme for Drupal 7 and 8. The styling it contains applies to the admin interface.
- ◆ Stark comes with no CSS whatsoever, and is designed to demonstrate the default HTML and CSS that's coming from Drupal.
- ◆ Stable exists as a fallback base theme if one hasn't been defined.
- ◆ Classy is a base theme designed to provide Drupal's standard classes, which you will be used to if you've worked with Drupal 7. Bartik is a sub-theme of Classy.

The description of Classy mentioned that it is a base theme. What does this mean? Base themes exist to give you a faster starting point for customizing your site's looks by doing some of the groundwork for you. They provide a standards-compliant starting point, some useful documentation, and really good reference for how to do certain theme tasks, such as creating new template files or adding new CSS. It is possible to chain multiple themes together if needed, to inherit features.

While some of these themes have existed in multiple versions of Drupal, there are distinct changes in the Drupal 8 theme system. These include semantic HTML5 markup, the Twig templating system (which we will look at later in this chapter), more JavaScript libraries included by default, and CSS classes being moved from preprocess into templating. Visit <https://www.drupal.org/node/2356951> for a more complete list.

There are many, many more themes you can download and make use of as a base theme too. Visit [https://www.drupal.org/project/project\\_theme](https://www.drupal.org/project/project_theme) to find a listing.

It's important to note that the themes in core are not there to be adjusted. By keeping them untouched, we can update them (if needed) in future without losing any work. So if we can't adjust them, how do we customize the look of our site?

## Time for action – creating a sub-theme

What we need to do is set up a theme to inherit the benefits we get from our chosen theme, that we can then adjust to suit our needs. This is called a **sub-theme**. Let's get started, using Bartik as the parent theme.

Back in PhpStorm, in the root directory of your project, navigate to the `themes` directory. Right now this directory is empty apart from a `README` file. We'll first create a directory to place our sub-theme in. It is good practice to place your own themes in a folder named `custom`, and any contributed themes in a sub folder named `contrib` (more on contributed themes later). Inside the newly created `custom` folder, we'll begin creating our sub-theme. We need to follow a few naming rules: it must start with a letter and only use lowercase alphanumerics and underscores. We're going to call ours `recipes`:

- 1.** Create a directory named `recipes`.
- 2.** Inside that, we'll define our theme. If you've worked with Drupal 7 previously, you may be used to working with the `.info` file. In Drupal 8, we now use the `.info.yml` files for theme definition. Among other things, the `.info.yml` file can define meta data, style sheets, and block regions. This is the only required file in the theme. Create the file, naming it `recipes.info.yml`, and then open it.
- 3.** Copy the following into your `recipes.info.yml` file and save:

```
name: Recipes
type: theme
base theme: bartik
description: A theme for styling up some delicious recipes!
core: 8.x
```

- 4.** Copy the file titled `logo.svg` from `/core/themes/bartik` into `/themes/recipes`.

- Now back on our site, click on the **Appearance** button in the shortcut bar. This will take you to `/admin/appearance`. This is where we can enable or disable the themes. Scroll down to the **Uninstalled themes** section and find our Recipes theme. Click on the **Install and set as default** button:

The screenshot shows the Drupal 8 administration interface. The top navigation bar includes 'Back to site', 'Manage', 'Shortcuts', and user information. Below it, the main menu has items like 'Content', 'Structure', 'Appearance' (which is highlighted), 'Extend', 'Configuration', 'People', 'Reports', and 'Help'. The 'Appearance' page displays the current theme configuration for 'Seven 8.0.0 (admin theme)'. It includes sections for 'GENERAL', 'CONTENT', 'DISPLAY', and 'DEVELOPMENT'. In the 'GENERAL' section, there's a link to 'Set as default'. Below this, the 'Uninstalled themes' section lists two themes: 'Recipes' and 'Stark 8.0.0'. The 'Recipes' theme card shows its preview, a brief description ('A theme for styling up some delicious recipes!'), and two buttons: 'Install' and 'Install and set as default'. The 'Install and set as default' button is circled in red. At the bottom of the page, under 'ADMINISTRATION THEME', it shows 'Administration theme' set to 'Seven'.

- Once the page has refreshed and notified you that Recipes is our default theme, click on the **Back to site** button in the top-left of your admin menu.

## What just happened?

We created a sub-theme that uses Bartik as its parent theme. But what's inside the Bartik theme? Next, we'll have a look at the inner workings of Bartik to understand how a theme is put together.

## An overview of Bartik

Let's have a look at what sits inside the Bartik directory and gain a bit of knowledge about what each file does (follow along by navigating to `core/themes/bartik`). The directory consists of:

- ◆ `bartik.info.yml`: Like our `recipes.info.yml` file, this defines some meta data required for the theme, such as its name, its parent theme, and a description. This file also describes some libraries, stylesheets, and theme regions. You can find information on the included regions and more on theme customization at <https://www.drupal.org/documentation/themes/bartik>.
- ◆ `bartik.libraries.yml`: This file defines the CSS libraries that can be loaded into the theme (it would also define the JavaScript, if Bartik had any by default). See the *Adding assets to your theme* section for more information and a practical example.
- ◆ `bartik.breakpoints.yml`: This includes the configuration for the Breakpoints module (<https://www.drupal.org/project.breakpoints>). This allows the theme to manage the breakpoints so that the other modules can make use of responsive breakpoints functionality. For example, Picture (<https://www.drupal.org/project/picture>) can deliver resized images depending on the size of the browser. See the *Mobile First, Responsive Themes* section for more information about the module
- ◆ `bartik.theme`: This contains some useful functions that support theming. These include adding classes to the markup detailing sidebar or region use, adding clearfix classes to some system elements, and classes to form elements.
- ◆ `logo.svg`: This is the stock logo that will show by default in the header. Logos can also be uploaded in `/appearance/settings`.
- ◆ `screenshot.png`: This displays on the `/admin/appearance` page and can be useful for quick theme recognition if toggling between themes.
- ◆ The `color` directory: This houses the preview functionality for Bartik's integration with the Color module. This module allows you to change the color scheme of the theme without making changes in the CSS.
- ◆ The `css` directory: This houses multiple modular CSS files that contain styling for all the components of the theme, such as buttons, forms, and the header. These are then either combined into libraries in `bartik.libraries.yml` or imported directly into `bartik.info.yml`, depending on their purpose.
- ◆ The `config` directory: This houses a schema for the Bartik configuration files.
- ◆ The `images` directory: This contains images used by the theme.

- ◆ The templates directory: This houses the templates used by the theme to structure the output of different areas of the site. The templates provide the HTML structure and some conditional logic. Templates target specific areas of the site which can range from fairly small independent sections like `status-messages.html.twig`, to elements that have wider effects, such as `page.html.twig`, which provides structure for all pages. You'll learn much more about templates and the language they're written in (Twig) later in the chapter.

We are now in a position to start making some changes to the default look that Bartik provides, so in the next section we'll add some custom CSS to test we're in business. Before getting to that, let's discuss responsive themes and mobile first principles.

## Mobile first, responsive themes

Drupal 8 themes are both mobile first and responsive out of the box. What do these terms mean and why are the themes set up this way?

In April of 2000, John Allsop wrote an article on <http://alistapart.com/> called *A Dao of Web Design*. This seminal article described an issue of emerging web design—designers and developers wanted to control their users' experience with as much rigidity as was possible in print. They wanted their sites to look as similar as possible in terms of functionality and layout, no matter the users' setup. The article advocated a fresh approach:

*"Make pages which are adaptable. Make pages which are accessible, regardless of the browser, platform or screen that your reader chooses or must use to access your pages."*

- John Allsop

This extends further today, as we must also consider different screen resolutions and Internet connection speeds, particularly for mobile devices. A site must adjust to the various needs of its users, and preferably, deliver an excellent experience, no matter the device.

Two approaches that attempted to solve the layout/screen size problem began to gain popularity: adaptive design and responsive design. Adaptive design was based on having series of static layouts that were delivered to the user after screen size was detected. Responsive design was based on fluidity of layout, responsively and flexibly fitting the device.

If a user resized their browser, an adaptive website would jump between the static layouts, while a responsive website would fluidly scale. Both approaches had benefits: adaptive design affords a level of control over the layout that sometimes can't be achieved with responsive design, but responsive design works more seamlessly across different devices, and because of this, has now become more popular.

Historically, designers and developers have focused on the desktop experience first, sometimes using layouts and functionality that didn't work well on mobile devices. This still happens occasionally, with designers approaching mobile design as a secondary issue. There's a very good reason to stamp this practice out: in 2015-2016, there are approximately 4.5 billion mobile phone users worldwide. A substantial portion of these will have Internet capability, and when we consider all the tablets, games consoles, watches, in-car devices, and even kitchen appliances with screens capable of browsing the Web, focusing on desktop experience alone is short sighted.

To combat the issue of reduced functionality of some devices, two approaches have become popular: graceful degradation and progressive enhancement. The result is the same; older or less capable devices deliver a lower level of experience, but importantly, a working site. However, the way this is achieved is applied from opposing starting points. Graceful degradation starts from a point of providing a high level of experience on modern devices, with the functionality degrading gracefully on older devices that aren't able to provide the support. Progressive enhancement takes the approach of creating the website at its lower level of user experience first, then progressively enhancing functionality if the devices support it. Both approaches have their merits, and excellent examples of both practices can be found at [https://www.w3.org/wiki/Graceful\\_degradation\\_versus\\_progressive\\_enhancement](https://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement).

The important point here is the ability to provide a working website, regardless of device, and we'll discuss how Drupal 8 can help you achieve this.

Mobile first is an approach that means that content is laid out intelligently on smaller devices. Content on a small screen tends to require a single column layout, and therefore a linear order, with the most important content at the top of the page. This isn't always the case, but it's a mindset that helps web professionals consider the best experience for smaller devices first, which tends to also lead to an improved large screen experience.

Drupal 8 can help you create a responsive, mobile first site with progressive enhancement or graceful degradation. First off, have a look at the homepage in your browser—try resizing the page, and look at how the elements adjust. At mobile, our admin menu shows our links with icons rather than text, to ensure the layout fits. Our site's main menu is sat within a clickable (or tappable) dropdown. The content is single column, with the sidebars sat underneath the main content area. As the browser expands, the admin menu includes text, the main menu drops out of the dropdown, and the sidebars are positioned on either side of the main content area. This is a good example of responsive, mobile first behavior. This is being achieved with CSS. As an example, look at the styling in **core | themes | bartik | css | components | header.css**. You can see the default styling being set at the top of the file, then further into the file, media queries appear, which provide the necessary changes for the layout to continue working well at larger sizes. If you're not familiar with media queries, you can find good documentation at [https://developer.mozilla.org/en-US/docs/Web/CSS/Media\\_Queries/Using\\_media\\_queries](https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries).

By providing good examples of responsive, mobile first styling in action, Drupal 8 has given you a helping hand beginning your own work.

Drupal 8 now comes with more JS libraries in core, which we'll discuss later. For now, just be aware that Modernizr is available to you, which is a great tool for detecting browser capabilities and providing methods for you to adjust your functionality and styling accordingly. See <https://modernizr.com/> for more information.

Drupal core also now ships with a module called **Breakpoint**. This allows you, on a theme by theme basis, to define the browser widths at which you want to be able to make functional changes. These settings are independent of any CSS media queries you might use. The Breakpoint settings help configure other modules, such as Responsive Images, another module now in core. The combination of these two modules allows you to switch images on your site. You can provide lower resolution images for small screen devices, then swap them for higher resolution for bigger screens. This is great for site performance, particularly important to bear in mind for mobile Internet users. The settings for breakpoints sit in the root of the theme. Bartik's can be found in `bartik.breakpoints.yml`.

We've now learned what the terms *mobile first* and *responsive* mean, and how Drupal 8 gives you the tools you need to begin following these practices.

Before we start using our theme, we're going to install a tool called **Drush**, which is a command line tool for Drupal. It'll help us move quickly through the next few sections.

## Time for action – installing Drush

The following section requires a command line with Unix or Unix-like functionality. If you are a Mac or Linux user, you'll have Bash preinstalled, but if you're a Windows user, you'll need to install Cygwin, which can be downloaded from <https://www.cygwin.com>. There are a number of tutorials available online to help you install and use it.

You'll also need some knowledge of editing and saving documents using terminal editors, such as `nano` or `vi`.



If you're new to command line, be aware that the dollar sign signals a new line of code (that is, don't type the dollar sign), and that each line needs to end with the return key being pressed.

We'll be installing Drush using Composer. If you have previous experience of Drupal 7 and already have Drush installed, upgrade it using your preferred method.

1. Firstly, we'll install Composer globally. In your terminal, run the following commands:

```
$ curl -sS https://getcomposer.org/installer | php  
$ mv composer.phar /usr/local/bin/composer
```

2. Now, open your `.bash_profile` (or `.zshrc` if using ZSH) using a terminal editor and add the following:

```
$ export PATH="$HOME/.composer/vendor/bin:$PATH"
```

3. Re-source the file (or reload your terminal):

```
$ source ~/.bash_profile
```

4. With Composer installed, we can quickly and easily install Drush:

```
$ composer global require drush/drush:dev-master
```



Composer makes it really easy to update Drush, or roll it back to a particular version if required. It's well documented online.

Drush has a lot of useful features. Visit <http://www.drush.org/en/master/> for documentation.

## Time for action – Adding assets to your theme

In Drupal 7, CSS and JS assets were added to themes via the `.info` file. In Drupal 8, asset management has been split out and we have the concept of an asset library. These libraries contain the CSS and JS we would like to attach. They can be applied globally or to a specific page. Let's start by creating a global-styling asset library:

1. In your theme's directory, create a file called `recipes.libraries.yml`.
2. Copy the following into your file and save:

```
global-styling:  
  version: 1.0  
  css:  
    theme:  
      css/style.css: {}
```



Indentation has meaning in `.yml` files, so ensure that the indentation in the preceding code matches your file.

3. We now need to attach our library to the site. This can be done in two ways. If we want to attach a library to a specific page, we can do that in our templates. We'll discuss this later in the chapter. But when we want to attach a library to all the pages on our site that use our theme, we can add it to our info file. In `recipes.info.yml`, copy and save the following at the bottom of the file:

```
- recipes/global-styling
```

Your info file should now look like this:

```
name: Recipes
type: theme
base theme: bartik
description: A theme for styling up some delicious recipes!
core: 8.x
libraries:
- recipes/global-styling
```

4. With our library in place and being applied globally via the info file, let's create some CSS and test it out. In your `recipes` directory, create a new directory called `css`.

5. In the `css` directory, create a file called `style.css`.

6. In `style.css`, copy and save the following:

```
.main-content h2 {
  font-size: 50px;
}
```

7. We now need to clear the cache. Drush makes this easy. Open your terminal and navigate to the site's root directory. We can type the following command to rebuild all our caches:

```
$ drush cr
```

This will do the equivalent of going to `/admin/config/development/performance` and clicking on the '**Clear all caches**' button, as shown in the following screenshot:

The screenshot shows the 'Performance' configuration page in the Drupal admin interface. The 'CLEAR CACHE' section contains a prominent 'Clear all caches' button, which is circled in red. Below it, the 'CACHING' section includes a note about the internal page cache module and settings for page cache maximum age (set to '<no caching>'). The 'BANDWIDTH OPTIMIZATION' section contains checkboxes for aggregate CSS and JavaScript files, both of which are checked. At the bottom is a 'Save configuration' button.

8. As the final step, let's head back to the homepage and see whether our styling has taken effect:

The screenshot shows the Drupal homepage displaying four content items. The first item is titled 'Interdico Paulatim Quidne'. The second item is titled 'Plaga Saluto Venio'. The third item is titled 'Awesome Sauce'. The fourth item is titled 'Haero Proprius Similis Verto'. Each content item has a 'Read more' link to its full details page. Red arrows point from the text labels 'Interdico Paulatim Quidne', 'Plaga Saluto Venio', 'Awesome Sauce', and 'Haero Proprius Similis Verto' to their respective content items on the page. The sidebar on the right lists other content items: 'Recipe List', 'Interdico Paulatim Quidne', 'Plaga Saluto Venio', 'Awesome Sauce', 'Haero Proprius Similis Verto', and 'Odio Pneum Ut Vel'.

## What just happened?

We added a CSS file by creating a new library and then made a styling adjustment to check that it worked.

Ok, our titles are bigger, so that tells us that our recipes theme is working. Excellent!

Next, let's make adjusting our assets as easy as possible while we're developing, by disabling one of Drupal's default performance features.

Drupal uses a site performance tool called aggregation. A site's performance can be affected by the addition of multiple CSS or JS files, each of which require an HTTP request to the server. What Drupal does to avoid this is condense multiple files into a single one, performing uglification and minification too. This means fewer HTTP requests are made, and the files that are requested are as small as possible. However, the aggregated files are not re-created every time you make a change to your CSS or JavaScript, as they are cached. Great for performance, but not so useful when we're developing. We could clear the cache each time we make a change, but a quicker way is to disable aggregation for now.

Let's go back to /admin/config/development/performance and uncheck the **Aggregate CSS files** and **Aggregate JavaScript files** options. Then, click on the **Save configuration** button, as shown in the following screenshot:

The screenshot shows the 'Performance' configuration page. At the top, there are sections for 'CLEAR CACHE' (with a 'Clear all caches' button) and 'CACHING' (with a note about the internal page cache module). In the 'BANDWIDTH OPTIMIZATION' section, two checkboxes are present: 'Aggregate CSS files' and 'Aggregate JavaScript files'. Both checkboxes are currently checked. A red oval highlights the 'Aggregate JavaScript files' checkbox. At the bottom of the page is a blue 'Save configuration' button.

With aggregation turned off, we can make changes and they will be immediately visible when we refresh our browser. Do remember to have aggregation enabled for production sites though!

## Time for action – calling assets on specific pages

Let's dive a little deeper with asset library dependencies. We're going to add a little piece of JavaScript with a dependency on jQuery; when a user clicks on the recipe description field, an alert box will be triggered to tell them it's delicious. We'll set it so that it only loads on the full view mode of our recipes (that is, not on our homepage listing). This will also be our first introduction to Twig. Let's begin:

1. Back in PhpStorm, in our `recipes` theme, create a directory called `js`.
2. In the `js` directory, create a file called `delicious.js`.
3. While the contents of this JS file are just to demo the concept, we'll create it and document it using good practices, so that you can use this as a basis for your own work. We'll start by defining the file, setting a description, and then creating our function:

```
/**  
 * @file  
 * A delicious notification alert  
 */  
var delicious = (function () {  
}());
```

4. After the function, we'll tell jQuery to load this function on the page load event:  
`jQuery(delicious.onPageLoad);`
5. We'll now start adding the contents of the function. We'll create the object that will be returned when the function is called inside our function:

```
/**  
 * Public object.  
 * @type {}  
 */  
var self = {};
```

6. Following our '`self`' object, we'll create our jQuery selector, picking up the content we want to bind a `click` function to:

```
/**  
 * The delicious recipe content.  
 *  
 * @type {jQuery}  
 */  
var $deliciousContent = jQuery('.delicious-content');
```

- 7.** Next, we'll bind the content click event and create our alerting function, which we'll make translatable:

```
/**  
 * Bind the recipe content click event.  
 */  
var bindRecipeContentEvent = function () {  
    $deliciousContent.on('click', function () {  
        deliciousAlert();  
    });  
};  
  
/**  
 * Alert the user that this is delicious.  
 */  
var deliciousAlert = function () {  
    window.alert(Drupal.t("This is delicious!"));  
};
```

- 8.** Finally, we'll set a property of our `self` object to be an anonymous function that calls the `bindRecipeContentEvent` function at page load. We'll then return the object:

```
/**  
 * Page load event.  
 */  
self.onPageLoad = function () {  
    bindRecipeContentEvent();  
};  
  
return self;
```

The contents of `delicious.js` should now be:

```
/**  
 * @file  
 * A delicious notification alert  
 */  
var delicious = (function () {  
    /**  
     * Public object.  
     * @type {}  
     */  
    var self = {};  
  
    /**  
     * The delicious recipe content.  
     * @type {jQuery}
```

```

        */
        var $deliciousContent = jQuery('.delicious-content');

        /**
         * Bind the recipe content click event.
         */
        var bindRecipeContentEvent = function () {
            $deliciousContent.on('click', function () {
                deliciousAlert();
            });
        };

        /**
         * Alert the user that this is delicious.
         */
        var deliciousAlert = function () {
            window.alert(Drupal.t("This is delicious!"));
        };

        /**
         * Page load event.
         */
        self.onPageLoad = function () {
            bindRecipeContentEvent();
        };

        return self;
    })();

jQuery(delicious.onPageLoad);

```

- 9.** With `delicious.js` in place, we now need to add it as an asset library. In PhpStorm, open `recipes.libraries.yml`. Underneath the global-styling asset library, we'll create another, called `delicious`. This one will include the JS we just created, and will also specify our dependency on `jQuery`, a version of which sits in core for us to use. (In Drupal 8, core contains a lot of libraries for dependencies, including `jQuery UI`, `Modernizr`, and `Backbone` to name a few. See `core/core.libraries.yml` for the complete list):

```

delicious:
  version: 1.0
  js:
    js/delicious.js: {}
  dependencies:
    - core/jquery

```

**10.** The last process is to add the asset library to the site. This time, we don't want it to apply globally, as we only want `delicious.js` and `jQuery` to load on the specific pages they're needed on. For that, we'll create a very specific Twig template. We're going to gloss over the details for now, as they'll be covered later in the chapter. In the `recipes` directory, create a new directory called `templates` and then inside that create one called `field`.

**11.** Inside `recipes | templates | field`, create a file named `field--node--field-description--recipe.html.twig`.

**12.** Inside that file, paste the following code. As mentioned, we'll gloss over Twig for now, but note that we're applying a class of `delicious-content` for our JS to apply to, and we're attaching the asset library using the `attach_library` function. This means that the asset library will only ever load on pages that include the specific recipe description field:

```
{%
set classes = [
'field',
'field--name-' ~ field_name|clean_class,
'field--type-' ~ field_type|clean_class,
'field--label-' ~ label_display,
'delicious-content',
]
%}
{%
set title_classes = [
'field_label',
label_display == 'visually_hidden' ? 'visually-hidden',
]
%}

{{ attach_library('recipes/delicious') }}

{%
if label_hidden %}
{%
if multiple %}
<div{{ attributes.addClass(classes, 'field_items') }}>
{%
for item in items %}
    <div{{ item.attributes.addClass('field_item') }}>{{ item.content }}</div>
{%
endfor %}
</div>
{%
else %}
{%
for item in items %}
    <div{{ attributes.addClass(classes, 'field_item') }}>{{ item.content }}</div>
{%
endfor %}
{%
endif %}
{%
endif %}
```

```

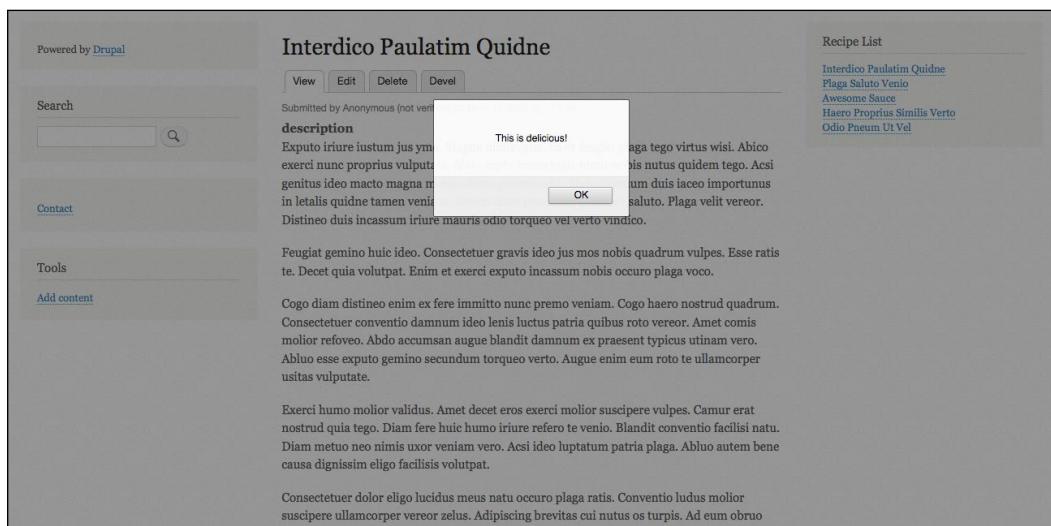
{ % endif %}
{ % else %}
<div{{ attributes.addClass(classes) }}>
  <div{{ title_attributes.addClass(title_classes) }}>{{
    label }}</div>
  { % if multiple %}
  <div class='field__items'>
    { % endif %}
    { % for item in items %}
      <div{{ item.attributes.addClass('field__item') }}>{{
        item.content }}</div>
    { % endfor %}
    { % if multiple %}
  </div>
  { % endif %}
</div>
{ % endif %}

```

### 13. Clear our caches with Drush:

```
$ drush cr
```

Ok, let's test that it's working! In your browser on the homepage, click on the title of one of the items in the list of recipes to be taken through to the full node. When you click on any of the text in the description section, our alert should appear:



Awesome! Now, let's test to see if `delicious.js` is loading only on our desired pages. If we view the source of the current page and do a search for it, we'll obviously find it:

```
/themes/recipes/js/delicious.js?v=1"></script>
```

Do the same search on the homepage and no results will be found. Exactly as we had hoped.

## ***What just happened?***

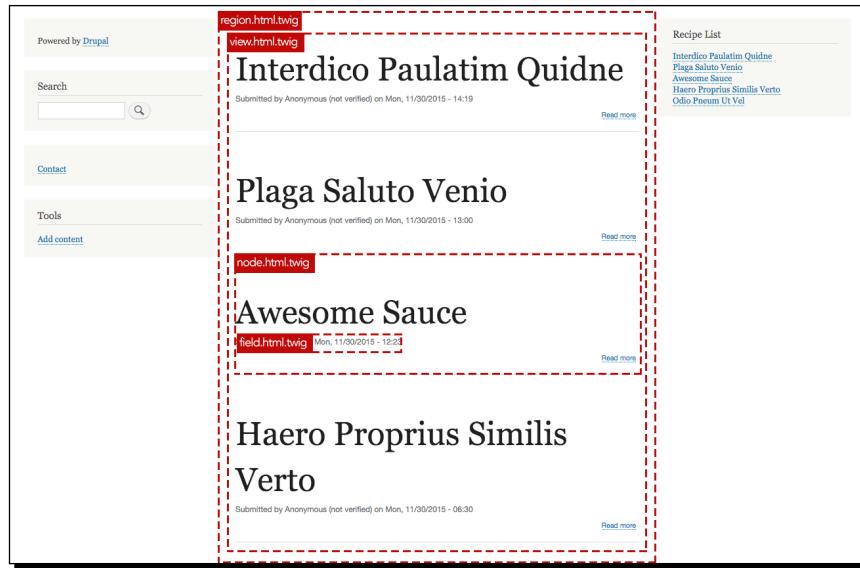
We created a simple piece of JS, added it as an asset library with a jQuery dependency, and used a Twig template to ensure that the asset library is only loaded on pages with the correct field.

## **Introduction to templating and Twig**

We skipped over templates and Twig in the previous section, so let's discuss them in more depth.

Templates will become the bread and butter of your experience of theming Drupal. They are the way we logically filter our data and create our required HTML structure. A standard page on a Drupal site is built from a set of nested templates. At each level of the nested structure, different variables are available to you that allow you to render the content you desire. Any of the templates mentioned next can be copied into your theme, either from Drupal core or your parent theme. The base versions of all the templates live in core, but your parent theme may already have templates you can copy that have a structure that gets you most of the way to your goal, or perhaps is already using a specifically named template you want to override.

Here's an example of some of the templates being used on our homepage:



A quick note for those moving from Drupal 7 to Drupal 8—you may be wondering about theme functions—they are no more. All themable output is run through templates.

The template nesting hierarchy starts with a template defining the basic structure of the HTML page, including the `<head>` and `<body>` tags. The base template that does this is `html.html.twig`, and is located in `core/modules/system/templates/html.html.twig`. It's well worth having a read of the comments provided in these base files for a description of what the template is for, as well as a list of the available variables.

After that, come the page templates. The base template for these is `page.html.twig`, located in `core/modules/system/templates/page.html.twig`. This is where elements such as `<header>`, `<main>`, and `<footer>` are defined.

It's very common to want to create specific instances of the page template; for example, a custom front page with a different structure to the default pages on the site. We do that by using theme hook suggestions.

## Theme hook suggestions

Theme hook suggestions are alternate templates you can create to override a default template file. They allow you to implement targeted overrides in your theme by using a specific naming convention.

When rendering an element, Drupal looks for template specificity—it uses the most specific template it can find. For example, when it's rendering your pages, if you create a template in your theme called `page-front.html.twig`, it uses that to render your front page. If no such file is found, it looks for the next most specific template, and continues to do that until it reaches the default `page.html.twig`.

In the next section, we'll move on to Twig debugging, where you will learn how to find out which templates your site is currently using, as well as discover some template naming suggestions for overriding them.

For further information and examples, refer to the following:

- ◆ [https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/function/hook\\_theme\\_suggestions\\_HOOK/8](https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/function/hook_theme_suggestions_HOOK/8)
- ◆ [https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/function/hook\\_theme\\_suggestions\\_alter/8](https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/function/hook_theme_suggestions_alter/8)
- ◆ [https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/function/hook\\_theme\\_suggestions\\_HOOK\\_alter/8](https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/function/hook_theme_suggestions_HOOK_alter/8)

For now, let's continue to look at templates and their hierarchy. After the `html.html.twig` template has defined the highest level `<html>`, `<head>`, and `<body>` tags, and `page.html.twig` has defined the `<header>`, `<main>`, and `<footer>` areas, the next level in our nested template tree are region templates. These are able to render the regions of the site (examples of regions in the Bartik theme are `sidebar_first`, `sidebar_second`, and `footer_first`, which are defined in its `.info.yml` file). The base region template, `region.html.twig` is located in `core/modules/system/templates/region.html.twig`.

Inside our regions, any number of nodes, blocks, or fields can be rendered. There's a lot of Drupal terminology to be learned, and if this is all sounding new to you, the best place to start is to have a thorough read of the template section of the theming guide located at <https://www.drupal.org/node/2186401>. There you'll find more information about template types, their base locations, and how to override them.

You may have noticed when we first looked at the core themes that there was one other directory there named `engines`. This houses the Twig templating engine, which replaces PHPTemplate from Drupal 7 as default (it is still possible, though not recommended, to use PHPTemplate). If you're new to Drupal, a templating engine, sometimes also called a theme engine, is the piece of software that combines our data with our theme's templates, and outputs HTML.

What was wrong with PHPTemplate? A number of things:

- ◆ Templates had too much power. For example, if a themer forgot to sanitize their user-submitted text, it left the site open to security issues (cross-site scripting). It was even possible to drop a site's database from a templating file! Feel like casually dropping the `users` table? `<?php db_query('DROP TABLE {users}'); ?>`
- ◆ It was complicated. Lots of templates and even more `theme()` functions. It had different syntaxes for rendering different data structures. All in all, a steep learning curve, which put a lot of developers off.
- ◆ Markup wasn't very pretty or human readable.
- ◆ It was Drupal-specific.

So why Twig?

- ◆ It's more secure. PHP functions, such as `db_query`, simply aren't available, and auto-escaping is enabled by default, so cross-site scripting is no longer an issue.
- ◆ The syntax is very easy to read and use. `<?php print render($content); ?>` becomes `{{ content }}`. More syntax examples to follow.
- ◆ Templates are extendable, thanks to its 'include' feature.
- ◆ Twig isn't Drupal-specific, so we learn transferable skills.
- ◆ It's well documented. See <http://twig.sensiolabs.org/documentation>.

Moving to Twig is probably the most radical frontend change in Drupal 8. Regardless of whether you're new to Drupal or you're making the move from Drupal 7, there will be some new material to learn. But as you'll see, the syntax is clean and human readable, and the effort you need to put into learning it is quickly recovered as you fly through writing your templates! Let's have a look at some of its features and syntax before we put it to use. If you're new to Drupal, the following section may be initially confusing, but should serve as a good reference point when you begin to work in Twig.

## File and function names

- ◆ PHPTemplate filename syntax: `block--search-form-block.tpl.php`
- ◆ Twig filename syntax: `block--search-form-block.html.twig`
- ◆ PHPTemplate function: `theme_node_links()`
- ◆ Twig file: `node-links.html.twig`

## Brackets syntax

Twig has three bracket types:

- ◆ {# #}: This is used for comments. Text inside these brackets never gets output.
- ◆ {{ }}: This is used for rendering. We'll see some examples in subsequent sections.
- ◆ {{ % }}: This is used for control structures such as `for` loops or conditionals. We'll see some examples in subsequent sections.

## Rendering

Rendering a variable is done by typing the variable name in the printing bracket syntax: {{ car }}

Let's use an example of a multidimensional `car` array in PHP:

```
$car = array(  
    'maxSpeed' => 200,  
    'gearbox' => 'six-speed',  
    'optionalUpgrades' => array(  
        'engine' => array(  
            'upgrade' => 'turbo-charged engine',  
            'price' => 500,  
        ),  
        'windows' => array(  
            'upgrade' => 'tinted windows',  
            'price' => 150  
        ),  
    ),  
);
```

To print an attribute of a variable, the powerful "dot" is used. {{ car.maxSpeed }} returns 200.

This dot syntax checks a number of possible scenarios for `maxSpeed`. The following is its process:

- ◆ Check if `car` is an array and `maxSpeed` a valid element
- ◆ If not, and if `foo` is an object, check that `maxSpeed` is a valid property
- ◆ If not, and if `foo` is an object, check that `maxSpeed` is a valid method (even if `maxSpeed` is the constructor—use `__construct()` instead)
- ◆ If not, and if `foo` is an object, check that `getMaxSpeed` is a valid method;
- ◆ If not, and if `foo` is an object, check that `isMaxSpeed` is a valid method;
- ◆ If not, return a null value

It's multidimensional: `{ { car.optionalUpgrades.engine.upgrade } }` returns turbo-charged engine.

At the time of writing, hashkeys in render arrays are not able to be targeted with the dot syntax. There is an active issue for this: <https://www.drupal.org/node/2160611> but for now, the syntax `[ '#haskey' ]` is needed, that is `{ { item.property['#markup'] } }`.

## Filters

With Twig, we can push a variable through a filter before it's rendered: `{ { car.gearbox | capitalize } }` returns Six-speed. As well as Twig's default filters, Drupal has included extras, such as translate: `{ { item | t } }`.

See a full list of the default Twig filters at <http://twig.sensiolabs.org/doc/filters/index.html>.

You can find the list of Drupal's extra filters in `core/lib/Drupal/Core/Template/TwigExtension.php` inside `getFilters()`.

## Control structures

The following are the different control structures:

- ◆ The if statements:

```
{% if car.maxSpeed %}  
    <div class="max-speed">This car will reach {{ car.maxSpeed }}  
    kph!</div>  
{% endif %}
```

- ◆ The for loops:

```
{% for item in optionalUpgrades %}  
    <li class="item-list upgrades"> {{ item.upgrade }} </li>  
{% endfor %}
```

- ◆ Variable creation:

```
{% set seatFinish = "leather" %}
```

- ◆ Array creation:

```
{% set colourScheme = [ 'blue', 'white', ] %}
```

## Debugging Twig

Twig, by default, offers a great debugging tool. With Drupal 8, it's extended even further. We'll enable it and begin using it in the next section, but for now, let's go through the abilities it gives us.

### HTML comments in markup

Drupal 8 extends the default Twig debug tool here, giving us information on template suggestions, as well as showing which template the markup is currently coming from:

```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'field' -->
<!-- FILE NAME SUGGESTIONS:
     * field--node--title--recipe.html.twig
     * field--node--title.html.twig
     * field--node--recipe.html.twig
     * field--title.html.twig
     * field--string.html.twig
     * field.html.twig
-->
<!-- BEGIN OUTPUT from 'core/themes/classy/templates/field/field--node--title.html.twig' -->
```

In the preceding screenshot, we can see the comments for the recipe title field on our homepage. We can see that currently the template it's using is coming from the Classy theme. We also have the template name suggestions if we need to create something more or less specific to suit our needs.

### Debugging variables

To print all the available variables in one of our template files, we can use the `dump()` function.

We can print all the available variables:

```
{ { dump() } }
```

There are a couple of global variables in Twig templates:

- ◆ `_context`: This contains all the variables that are passed to the template file, such as variables prepared by preprocess, coming from `theme()`, or set from within the template
- ◆ `{ { dump() } }`: This is equivalent to `{ { dump(_context) } }`
- ◆ `{ { dump(\_context|keys) } }`: To print available keys
- ◆ `_charset`: This references the (current) charset

But beware, `{ { dump() } }` may result in a white screen of death. The process recursively moves through and prints a potentially very large list of variables, which can result in a loss of memory. If that's the case, you can do something like the following:

```
<ol>
  { % for key, value in _context %}
    <li>{{ key }}</li>
  { % endfor %}
</ol>
```

To print the content of a specific variable, like `$car`:

```
{ { dump($car) } }
```

To print out the content of the `$car` and `$truck` variables:

```
{ { dump($car, $truck) } }
```

## Kint

`{ { dump() } }` is an excellent tool, but if the element it's applied to contains a lot of information, the output can be difficult to interpret. The Devel module, which was used to create the dummy content for the site, contains a submodule called Kint. This module can be enabled with Drush:

```
$drush en kint
```

With it enabled, all the `{ { dump() } }` commands can be replaced with `{ { kint() } }`, which gives a more user-friendly output.

## Time for action – Twig in practice

Ok, let's put some of what you've learned in the previous section into practice. It would be nice to separate out the preparation stage in our recipe from the cooking stage. We'll do this by splitting the two types of fields into separate wrapping `div` tags, and then apply some styling to help us differentiate them:

1. We'll find out which template is currently in use by enabling Twig debugging. In PHPStorm, navigate to **sites / default**. If you have a file named `services.yml`, open it. If you don't, make a copy of `default.services.yml`, rename it to `services.yml`, and open it.
2. Look for the section called `twig.config`. In this section, you will find an option that currently says `debug: false`. Change that value to `true` and save. Like aggregation for CSS and JS, this will improve your development conditions, but ensure it's turned off again before the site is in production.

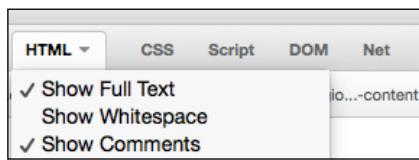
3. If in step 3 you encountered a permissions error, you will need to reset permissions in the **sites / default** directory. In terminal, navigate to the sites directory and then enter the following:

```
$ chmod 755 default
```

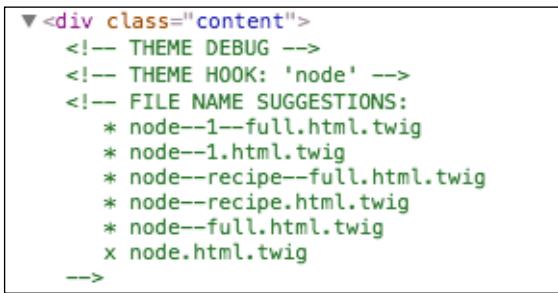
4. We'll also enable Kint to help us with some debugging:

```
$ drush en kint
```

With debugging enabled, let's have a look at the HTML output mentioned earlier. In your browser, click through to a full recipe page (I'm going to use the Awesome Sauce recipe, and I've added an image into the image field), and then inspect the HTML with your inspector of choice. (Right-click or *cmd + click* on some of the content, then, depending on your browser and tools, you'll find an **Inspect** option.) Note: if you're using Firebug, ensure comments are turned on:



There's a lot of information available to you at this point, but you'll soon find that you can make your way through it with relative ease. Find the `<article>` element with a class of `node--type--recipe`. This is the structural point at which all the elements are wrapped in one containing div, so this is what we need to split:



The template information regarding the element sits above it. As we can see from the green comments, the template being used is currently in `core/themes/bartik/templates/node.html.twig`. We can also see a number of file name suggestions, which we can choose from for extra specificity. We'll make use of the `node--recipe--full.html.twig` suggestion, which as the name implies, only applies to the full view mode.

5. In PHPStorm, inside your theme's template directory, create a directory called content. Note that the directory names do not affect Drupal's ability to find a template. They are there to assist the developer in keeping work organized. Inside the content directory, create a file called node--recipe--full.html.twig.
6. We'll now copy the contents of core/themes/bartik/templates/node.html.twig over into our newly created node--recipe--full.html.twig, so that we can begin overriding it. After that, your file's content should look as follows (the length comment is excluded):

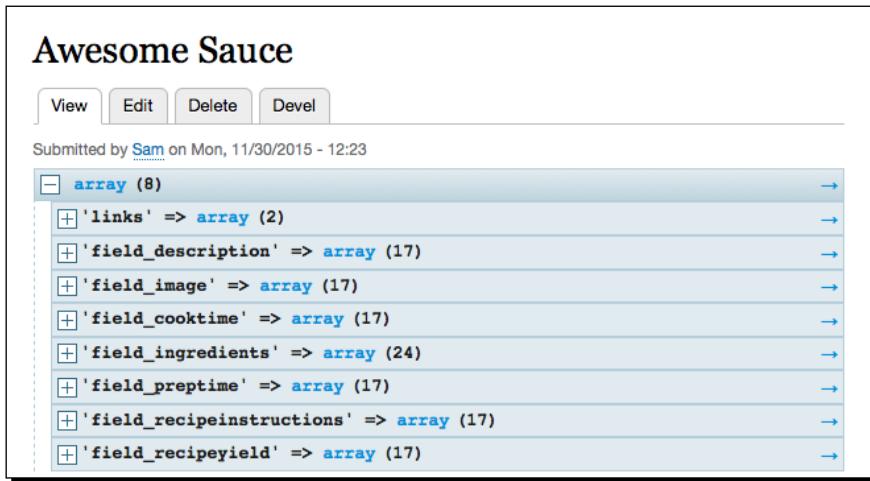
```

{#
{%
set classes = [
'node',
'node--type-' ~ node.bundle|clean_class,
node.isPromoted() ? 'node--promoted',
node.isSticky() ? 'node--sticky',
not node.isPublished() ? 'node--unpublished',
view_mode ? 'node--view-mode-' ~ view_mode|clean_class,
'clearfix',
]
%}
{{ attach_library('classy/node') }}
<article{{ attributes.addClass(classes) }}>
    <header>
        {{ title_prefix }}
        {% if not page %}
            <h2{{ title_attributes.addClass('node__title') }}>
                <a href="{{ url }}" rel="bookmark">{{ label
                }}</a>
            </h2>
        {% endif %}
        {{ title_suffix }}
        {% if display_submitted %}
            <div class="node__meta">
                {{ author_picture }}
                <span{{ author_attributes }}>
                    {% trans %}Submitted by {{ author_name }} on {{ date
                    }}{% endtrans %}
                </span>
                {{ metadata }}
            </div>
        {% endif %}
    </header>
    <div{{ content_attributes.addClass('node__content',
        'clearfix') }}>
        {{ content }}
    </div>
</article>

```

Before doing anything to the file, see if you can spot some of what we've been discussing: array creation, conditional logic, and variable rendering.

7. What we're interested in is the `{ { content } }` variable, as we'll need to drill into this to find the different fields we want to split into our two separate wrapper `div` tags. We'll use Kint to debug its contents. Adjust `{ { content } }` to `{ { kint(content) } }`, and then clear your cache with:  
`$ drush cr.`
8. When you reload your page, the content will have disappeared (because it's no longer being rendered), and in its place you will have the debugging information. Open the array and then close down each inner array item (which will have opened by default), so that you're left with the following:



The screenshot shows a Drupal page titled "Awesome Sauce" with a "View" button. Below the title are four buttons: "View", "Edit", "Delete", and "Devel". A timestamp "Submitted by Sam on Mon, 11/30/2015 - 12:23" is displayed. The main content area contains a Kint dump of an array. The array has 8 items, each with a plus sign and a key like 'links' or 'field\_description'. To the right of each item is a blue arrow pointing right.

```
[+]'links' => array (2)
[+]'field_description' => array (17)
[+]'field_image' => array (17)
[+]'field_cooktime' => array (17)
[+]'field_ingredients' => array (24)
[+]'field_preptime' => array (17)
[+]'field_recipeinstructions' => array (17)
[+]'field_recipeyield' => array (17)
```

We can see that the content array contains eight items, seven of which have keys with names of the different fields on our page. We'll be using these to print our custom output.

9. There are some fields that are neither preparation nor cooking-based, and these ones we'd like to stay rendered in the normal `div`. Fields such as the description and the image. We want to exclude everything else though, so we can print them separately. We're going to make use of a Drupal Twig filter called `without`, which renders an element without specific inner items. Back in the template, replace `{ { kint(content) } }` with `{ { content|without('field_cooktime', 'field_preptime', 'field_ingredients', 'field_recipeinstructions') } }`. Our page will now only include the items we still want rendered in the normal way:



- 10.** Now we'll render the fields in separate `div` tags. Directly beneath that last Twig statement, paste the following:

```
<div class="preparation recipe-phase">
  {{ content.field_prepetime }}
  {{ content.field_ingredients }}
</div>
<div class="cooking recipe-phase">
  {{ content.field_cooktime }}
  {{ content.field_recipeinstructions }}
</div>
```

- 11.** Now we'll go back to the theme's `style.css` file in the `css` directory and add some styling. Paste and save the following underneath the existing styling:

```
/* Override default image float on the recipe node */
.node--type-recipe .field--name-field-image {
  float: none;
}

.recipe-phase {
  padding: 20px;
  margin-bottom: 20px;
}

.preparation {
  background-color: #e7e7e7;
```

```
.cooking {  
    background-color: #b0fbad;  
}
```

Let's have a look at the page again:

Powered by [Drupal](#)

## Awesome Sauce

[View](#) [Edit](#) [Delete](#) [Devel](#)

Submitted by [Sam](#) on Mon, 11/30/2015 - 12:23

**description**  
A deliciously sweet and spicy sauce that makes everything you put it on that much awesome.  
A little goes a long way...

[Contact](#)

[Tools](#)

[Add content](#)

**Image**



**recipeYield**  
12 Servings

**prepTime**  
1/6 hours

**ingredients**

One ghost pepper (optional)  
Two habanera peppers  
Three Thai peppers  
Four jalapeno peppers  
Four garlic cloves  
Three cups of rice vinegar  
One tea spoon of fish sauce  
One cup of sugar

**cookTime**  
1/2 hours

**recipeInstructions**

1. Remove the stems from the peppers.
2. Add the peppers and garlic to a food processor, and blend until pureed.
3. Add vinegar, sugar, fish sauce, and puree to a small saucepan, and bring to a simmer over low heat.
4. Simmer sauce for 20 to 30 minutes, until the sugar has completely dissolved.
5. Remove the saucepan from the burner, and let stand for 10 minutes.
6. Your Awesome Sauce is ready to serve, or it can be refrigerated for up to three weeks.

Thai peppers and fish sauce are typically available in most Asian markets. Ghost peppers are typically considered to be the hottest pepper in the world, and may be left out for those that have a little less tolerance for heat, or if you aren't able to find them.

Recipe List

[Interdico Paulatim Quidne](#)  
[Plaga Saluto Venio](#)  
[Awesome Sauce](#)  
[Haero Proprius Similis Vertu](#)  
[Odio Pneum Ut Vel](#)

## **What just happened?**

We used Twig debugging with Kint to determine the template we needed, and what variables were available to us. We used a Drupal-specific Twig filter to render content minus some items, and rendered those separately in the customized structure we wanted.

We got the results we wanted with this method, but it required us to break arrays apart. For larger content types with more fields, this would become cumbersome. There's another method, requiring the use of a contributed module. Let's take a look at that.

## **Time for action – understanding the benefits of contributed modules**

Let's try to achieve the same result we got in the previous section, but this time, by using a contributed module called `field_group`:

- 1.** Delete `node--recipe--full.html.twig` inside **recipes / templates / content** (or move the file outside of the project if you'd prefer to keep a copy of the work) and clear your cache.
- 2.** In command line, navigate to anywhere inside your Drupal project and type the following:

```
$ drush en field_group -y
```

This command will enable the `field_group` module if you have it downloaded. If not, it will go ahead and download the latest stable release for you and enable it.

3. Now back in the browser, navigate to `/admin/structure/types/manage/recipe/display` where the display settings for the recipe content type exist. You will see that we have a new piece of functionality at the bottom of the page:

The screenshot shows the 'Manage display' configuration for the 'recipe' content type. At the bottom of the list of fields, there is a section titled 'Add new group'. This section contains three input fields: 'Label' (with the value 'group\_'), 'group name (a-z, 0-9, \_)' (with the value 'group\_'), and a dropdown menu labeled 'Fieldset'. A red box highlights this entire 'Add new group' section.

4. Type `preparation` into both the text inputs, select **Html element** from the dropdown, and hit **Save**.
5. When the page reloads, you'll see the preparation group is now placed above the **Disabled** label. On the right-hand side of the preparation group item, there's a cog. Click on that.
6. The settings for the group will now open. Leave everything as default, apart from the **Extra CSS classes** text input, where you can type `preparation recipe-phase`, and then click on the **Update** button.

Field group format: html\_element

**Field group label**  
preparation

**Element**  
div

E.g. div, section, aside etc.

**Show label**  
No

**Label element**  
h3

**Attributes**

E.g. name="anchor"

**Effect**  
None

**Speed**  
Fast

**ID**

**Extra CSS classes**  
preparation recipe-phase

**Update** **Cancel**

7. Replicate these steps again, this time swapping preparation with cooking. When finished, you should have the two groups enabled.
8. Now for some drag-and-dropping. Drag both the group fields up and place them underneath the **recipeYield** field. Then, grab the **prepTime** field and drag it above the preparation group item. You'll see that the icon automatically shifts to the right, as if it's indented. Drop that field there and then drag the **ingredients** field into the preparation item area too, dropping it underneath **prepTime**. Then drag **cookTime** and **recipeInstructions** into the cooking group area. When finished, your items should look like this:

|                       |              |
|-----------------------|--------------|
| ⊕ Links               | Visible      |
| ⊕ description         | Above        |
| ⊕ Image               | Above        |
| ⊕ recipeYield*        | Above        |
| ⊕ preparation*        | Html element |
| ⊕ prepTime*           | Above        |
| ⊕ ingredients*        | Above        |
| ⊕ cooking*            | Html element |
| ⊕ cookTime*           | Above        |
| ⊕ recipeInstructions* | Above        |

Visible

Default

Image

Original image

Plain text

Html element

Duration

Plain text

Duration

Default

Extra CSS classes: preparation recipe-phase  
Element: div

delete

Extra CSS classes: cooking recipe-phase  
Element: div

delete

- 9.** Click on the **Save** button. Once the page has refreshed, you can see a notification as the following:

The screenshot shows the 'Manage display' configuration page for a 'Recipe' content type. At the top, there are tabs for 'Edit', 'Manage fields', 'Manage form display', 'Manage display' (which is selected), 'Translate content type', and 'Devel'. Below the tabs, there are two view modes: 'Default' and 'Teaser'. A red arrow points from the text 'Your settings have been saved.' in a green success message box at the bottom left to the 'Save' button in the top right corner of the page.

- 10.** Click on the **Back to site** button in the top-left of your admin menu. We have achieved the same result:

The screenshot shows the final theme result for the 'Awesome Sauce' recipe. The page includes a header with 'Powered by Drupal', a search bar, and links for 'Contact' and 'Tools'. The main content area features the title 'Awesome Sauce', a small image of a bowl of sauce, and a detailed list of ingredients and instructions. A sidebar on the right lists other recipes. The entire page is styled with a clean, modern look using light colors and clear typography.

## **What just happened?**

We removed the templating we did with Twig and then replicated the functionality with the `field_group` module.

It's pretty neat that we were able to achieve the same amount of control over our markup so quickly and easily. What's more, this method is achievable by site builders as well as developers, and it will continue to be a quicker method for updating the content type if more fields are added. With this being the case, how should we decide whether to use a contributed module on our websites?

## **What are contributed modules?**

Contributed modules, or contrib modules, are pieces of Drupal functionality, contributed to the community and maintained by the community of developers, sometimes sponsored by companies keen to support open source. You can find all contrib modules in the modules section of Drupal at [https://www.drupal.org/project/project\\_module](https://www.drupal.org/project/project_module). The `field_group` module can be found at [https://www.drupal.org/project/field\\_group](https://www.drupal.org/project/field_group).

## **How do I know whether a module is safe to use?**

It would indeed be risky to randomly pick and choose modules to use without prior knowledge, but there's lots of information about modules on Drupal (<https://www.drupal.org/>) that can inform you before use. These include:

- ◆ The number of people maintaining the project. The more eyes on the code, the more likely it is to be good quality.
- ◆ The number of commits made, and how recently. Active maintenance means the module is more likely to be protected against any recent security issues.
- ◆ The issue and bug queue. It's less about the number of bugs or issues and more about how well they're responded to.
- ◆ The number of modules that depend on it. If a module is excellent, more and more developers will feel confident in having it as a dependency. A lot of core modules started life this way.
- ◆ Whether the module is well documented.
- ◆ The number of downloads and reported installations. Again, the more eyes on it, the better, and the more users, the greater the likelihood that bugs are found early.

All of this is available to explore on a module's Drupal (<https://www.drupal.org/>) page.

## Is it better to use a contrib module or custom code?

There is no hard and fast answer to this. The benefits of a contrib module include the following:

- ◆ Security. If lots of other developers are using it, it can give you confidence in code quality and security fixes.
- ◆ You're not required for maintenance (unless you choose to contribute back to it).
- ◆ Ease of use. As with the `field_group` example, it allows people with less development expertise to configure a powerful website.
- ◆ A good learning opportunity. Dive into the module and explore how it is coded.
- ◆ Usually more scalable.

The benefits of custom code include:

- ◆ A clean, non-bloated solution to a unique problem.
- ◆ Site performance. Large contrib modules with lots of extra functionality will negatively affect a site's performance more than clean, well-written code that addresses a single issue.
- ◆ It gives you an opportunity to improve and test your development skills.

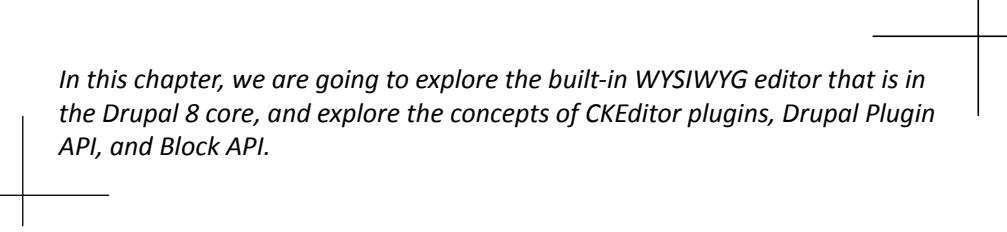
## Summary

In this chapter, we explored theming in Drupal 8, from what a theme is and how to create one, to loading and using CSS and JS assets, as well as templating, and how to determine when and why to make use of contributed modules.

In the next chapter, we will explore CKEditor and be introduced to Block API for Drupal 8.

# 6

## Enhancing the Content Author's User Experience



*In this chapter, we are going to explore the built-in WYSIWYG editor that is in the Drupal 8 core, and explore the concepts of CKEditor plugins, Drupal Plugin API, and Block API.*

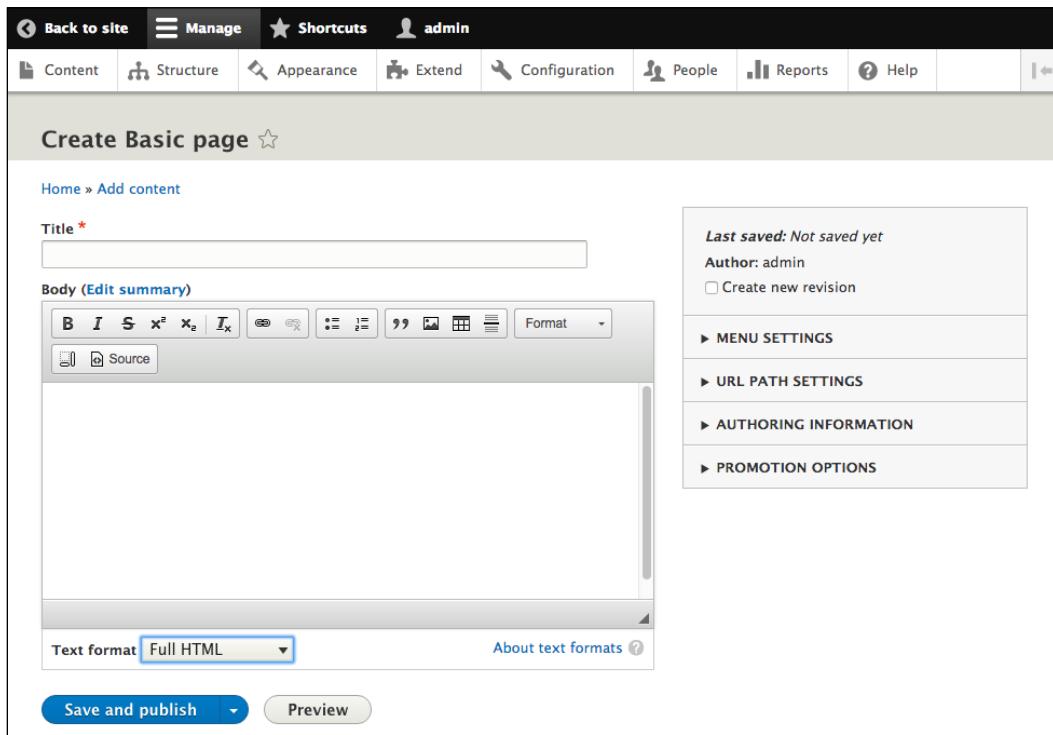
In this chapter, we will learn:

- ◆ How to use the built-in WYSIWYG editor for Drupal
- ◆ How inline editing works and how to get the best out of it
- ◆ How to make changes and save them through configuration
- ◆ How to create a custom block via the new Block API
- ◆ How to include default configuration in our custom block

### A quick introduction to CKEditor in Drupal 8

Drupal 8 is finally shipping with a default WYSIWYG editor: CKEditor. CKEditor is an open source text editor that aims to bring word processing features to web pages. In previous versions of Drupal, several modules tried to fill this gap, but their configuration at most times was a bit tricky as they were relying on external JavaScript libraries.

Bringing CKEditor's capabilities directly in core, Drupal 8 can provide a richer authoring experience to content editors. Users are given a drag-and-drop interface to customize available functionality and export configurations to share with other systems. Developers are given a unified way of accessing properties, adding plugins, and extending the editor's functionality:



## Configuring CKEditor profiles

CKEditor profiles would not be much without the ability of being customized to the editor's liking. Now we are going to customize the basic HTML profile. Log in to your Drupal 8 website and you should see the new admin toolbar.

### Time for action - adding some buttons to the basic HTML profile

1. Click on Configuration and then on Text formats and editors:

The screenshot shows the Drupal 8 Configuration page. At the top, there is a navigation bar with links for Back to site, Manage, Shortcuts, and admin. Below the navigation bar, the title 'Configuration' is displayed. Under the 'Configuration' title, there is a message box stating 'One or more problems were detected with your Drupal installation. Check the status report for more information.' Below the message box, there are several sections: 'PEOPLE' (Account settings), 'CONTENT AUTHORIZING' (Text formats and editors), 'SYSTEM' (Site information, Cron), and 'USER INTERFACE' (Shortcuts). The 'Text formats and editors' link under 'CONTENT AUTHORIZING' is highlighted in blue, indicating it is the current section being viewed.

- 2.** On the **Text formats and editor** screen, click on the **Configure** button next to the **Basic HTML** profile:

The screenshot shows the 'Text formats and editors' configuration page. At the top, there are navigation links: Back to site, Manage, Shortcuts, and admin. Below that is a toolbar with icons for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. The main title is 'Text formats and editors'. A breadcrumb trail shows: Home > Administration > Configuration > Content authoring. A note below the title states: 'Text formats define how text is filtered for output and how HTML tags and other text is displayed, replaced, or removed. Improper text format configuration is a security risk. Learn more on the Filter module help page.' Another note says: 'Text formats are presented on content editing pages in the order defined on this page. The first format available to a user will be selected by default.' A 'Show row weights' link is visible above the table. The table has columns: NAME, TEXT EDITOR, ROLES, and OPERATIONS. It lists four formats: Basic HTML (CKEditor, Authenticated user, Administrator), Restricted HTML (—, Anonymous user, Administrator), Full HTML (CKEditor, Administrator), and Plain text (—, note: 'This format is shown when no other formats are available'). Each row has a 'Configure' button in the OPERATIONS column. A 'Save changes' button is at the bottom left of the table area.

| NAME            | TEXT EDITOR | ROLES   | OPERATIONS                |
|-----------------|-------------|---|---------------------------|
| Basic HTML      | CKEditor    | Authenticated user,Administrator                                | <a href="#">Configure</a> |
| Restricted HTML | —           | Anonymous user,Administrator                                    | <a href="#">Configure</a> |
| Full HTML       | CKEditor    | Administrator   | <a href="#">Configure</a> |
| Plain text      | —           | <i>This format is shown when no other formats are available</i> | <a href="#">Configure</a> |

[Save changes](#)

3. Scroll down to **TOOLBAR CONFIGURATION**, and drag the image icon out of the toolbar into the pool of other elements to remove it from the configuration. Note that the image and the configuration are no longer part of the page:

The screenshot shows the CKEditor plugin settings page in the Drupal admin interface. At the top, there are tabs for Back to site, Manage, Shortcuts, and admin. Below the tabs, the page title is "Basic HTML" and the machine name is "basic\_html".  
**Roles**:  
- Anonymous user (unchecked)  
- Authenticated user (checked)  
- Administrator (checked)  
**Text editor**: CKEditor  
**TOOLBAR CONFIGURATION**:  
Move a button into the *Active toolbar* to enable it, or into the list of *Available buttons* to disable it. Buttons may be moved with the mouse or keyboard arrow keys. Toolbar group names are provided to support screen reader users. Empty toolbar groups will be removed upon save.  
**Available buttons**: A grid of icons representing various toolbar functions.  
**Active toolbar**: A horizontal toolbar with icons for bold, italic, underline, etc., followed by a "Format" dropdown.  
**Show group names**: A link to show toolbar group names.  
**CKEditor plugin settings**:  
**Image**:  
- Uploads enabled, max size: 2 MB  
- Enable image uploads (checked)  
- Upload directory: inline-images  
A directory relative to Drupal's files directory where uploaded images will be stored.  
- Maximum file size: 2 MB  
If this is left empty, then the file size will be limited by the PHP maximum upload size of 2 MB.  
- Maximum dimensions:  
width [ ] x height [ ] pixels  
Images larger than these dimensions will be scaled down.

4. Click on **Save configuration** at the very bottom of the page:

The screenshot shows the 'Filter processing order' configuration page. At the top right is a 'Show row weights' link. Below it is a list of filters:

- Limit allowed HTML tags and correct faulty HTML
- Align images
- Caption images
- Restrict images to this site
- Track images uploaded via a Text Editor

Below this is a 'Filter settings' section for the 'Limit allowed HTML tags and correct faulty HTML' filter. It is set to 'Enabled'. A detailed description of the allowed HTML tags follows:

**Allowed HTML tags**

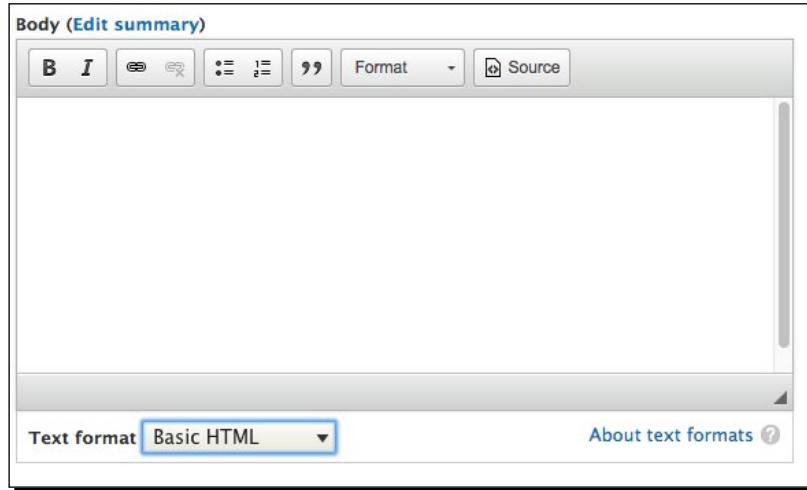
```
<a href hreflang> <em> <strong> <cite> <blockquote cite> <code> <ul type> <ol start>
```

A list of HTML tags that can be used. By default only the `lang` and `dir` attributes are allowed for all HTML tags. Each HTML tag may have attributes which are treated as allowed attribute names for that HTML tag. Each attribute may allow all values, or only allow specific values. Attribute names or values may be written as a prefix and wildcard like `jump-*`. JavaScript event attributes, JavaScript URLs, and CSS are always stripped.

Display basic HTML help in long filter tips  
 Add `rel="nofollow"` to all links

At the bottom is a blue 'Save configuration' button.

5. That's it! If you visit a content page form and choose the **Basic HTML** profile, then the image button will not be there anymore:



## What just happened?

Following this procedure, we customized the existing profiles by removing the upload image button from the CKEditor profile.

Configurations are exportable in D8 and can be imported to other environments.

### Time for action - exporting CKEditor configuration

1. Click on **Configuration**. Then click on **Configuration synchronization** and select the **Export** tab. Click on **Single item**.
2. On the **Configuration type**, choose **Text Editor**, and on **Configuration name**, choose **Basic HTML**:

The screenshot shows the 'Single export' page in the Drupal 8 admin interface. The top navigation bar includes 'Back to site', 'Manage', 'Shortcuts', and 'admin'. Below the navigation is a toolbar with links for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. The main content area is titled 'Single export' with a star icon. A navigation bar at the top of the content area has tabs for 'Synchronize', 'Import', and 'Export', with 'Export' being the active tab. Below this is another tab bar with 'Full archive' and 'Single item', with 'Single item' being the active tab. The URL in the browser is 'Home > Administration > Configuration > Development > Synchronize'. The page content is titled 'Here is your configuration:' and contains YAML code for the 'Basic HTML' configuration. The configuration details are as follows:

```

uid: 4075a713-3d0e-4e29-a261-9726ee17cb91
langcode: en
status: true
dependencies:
  config:
    - filter.format.basic_html
  module:
    - ckeditor
format: basic_html
editor: ckeditor
settings:
  toolbar:
    rows:
      -
      -
        name: Formatting
        items:
          - Bold
          - Italic
      -
        name: Linking
        items:
          - DrupalLink
          - DrupalUnlink
  
```

Filename: `editor.editor.basic_html.yml`

- 3.** Copy the configuration and import it into another environment by clicking on **Configuration**. Then click on **Configuration synchronization** and select the **Import** tab. Click on **Single item**:

The screenshot shows the 'Single import' page within the 'Configuration synchronization' module. The top navigation bar includes links for Home, Manage, Shortcuts, and admin. Below the navigation is a toolbar with icons for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. The main title 'Single import' is displayed with a star icon. A navigation bar below the title has tabs for 'Synchronize', 'Import' (which is selected), and 'Export'. Underneath the tabs are buttons for 'Full archive' and 'Single item'. The main content area shows the URL 'Home » Administration » Configuration » Development » Synchronize'. It instructs the user to 'Import a single configuration item by pasting its YAML structure into the text field.' A dropdown menu labeled 'Configuration type \*' is set to 'Text Editor'. Below it is a large text input field with the placeholder 'Paste your configuration here \*'. At the bottom left of the input field is a link '► ADVANCED'. At the very bottom is a blue 'Import' button.

## **What just happened?**

You have configured an existing profile, removed some fields, and learned how to export that configuration and import it into another environment.

## Adding a new CKEditor profile

Most often, built-in profiles cover a website's needs. The defaults, however, do not provide a way for anonymous users to have a smooth editing experience when inputting text (for example, comments). To provide that functionality, we need to create a new profile and assign it to the anonymous user.

### Time for action - creating a text-only control profile for anonymous users

- From the admin toolbar, go to **Configuration | Text formats and editors** and choose **Add text format**:

| NAME            | TEXT EDITOR | ROLES   | OPERATIONS                 |
|-----------------|-------------|---|----------------------------|
| Basic HTML      | CKEditor    | Authenticated user,Administrator                                | <button>Configure</button> |
| Restricted HTML | —           | Anonymous user,Administrator                                    | <button>Configure</button> |
| Full HTML       | CKEditor    | Administrator   | <button>Configure</button> |
| Plain text      | —           | <i>This format is shown when no other formats are available</i> | <button>Configure</button> |

- For the name, enter **Text only controls**. Tick on **Anonymous user** and choose **CKEditor** in the **Text editor** drop-down box.
- Add the **Underline** button and keep **Bold**, **Italic** and **Ordered/Unordered lists**.
- Enable the following filters:
  - Limit allowed HTML tags and correct faulty HTML**
  - Convert line breaks into HTML (i.e. <br> and <p>)**
  - Correct faulty and chopped off HTML**

## 5. Save the configuration:

**Name \***  
Text only controls

**Roles**  
 Anonymous user  
 Authenticated user  
 Administrator

**Text editor**  
CKEditor

**TOOLBAR CONFIGURATION**  
Move a button into the *Active toolbar* to enable it, or into the list of *Available buttons* to disable it. Buttons may be moved with the mouse or keyboard arrow keys. Toolbar group names are provided to support screen reader users. Empty toolbar groups will be removed upon save.

**Available buttons**

**Active toolbar**

**Enabled filters**

- Limit allowed HTML tags and correct faulty HTML
- Display any HTML as plain text
- Track images uploaded via a Text Editor
 

Ensures that the latest versions of images uploaded via a Text Editor are displayed.
- Align images
 

Uses a `data-align` attribute on `<img>` tags to align images.
- Convert line breaks into HTML (i.e. `<br>` and `<p>`)
- Caption images
 

Uses a `data-caption` attribute on `<img>` tags to caption images.
- Convert URLs into links
- Restrict images to this site
 

Disallows usage of `<img>` tag sources that are not hosted on this site by replacing them with a placeholder image.
- Correct faulty and chopped off HTML

**Filter processing order**

- + Limit allowed HTML tags and correct faulty HTML
- + Convert line breaks into HTML (i.e. `<br>` and `<p>`)
- + Correct faulty and chopped off HTML

**Filter settings**

**Limit allowed HTML tags and correct faulty HTML**  
Enabled

**Allowed HTML tags**

<a href hreflang> <em> <strong> <cite> <blockquote cite> <code> <ul type> <ol start>

A list of HTML tags that can be used. By default only the `lang` and `dir` attributes are allowed for all HTML tags. Each HTML tag may have attributes which are treated as allowed attribute names for that HTML tag. Each attribute may allow all values, or only allow specific values. Attribute names or values may be written as a prefix and wildcard like `jump-`. JavaScript event attributes, JavaScript URLs, and CSS are always stripped.

Based on the text editor configuration, these tags have automatically been added: <u>.

Display basic HTML help in long filter tips  
 Add `rel="nofollow"` to all links

**Save configuration**

6. The **Text formats and editors** configuration page should contain the newly created format. Move the new format above the **Restricted HTML** format, which is an alternative provided by Drupal for anonymous users skipping the editor altogether:

| NAME               | TEXT EDITOR | ROLES   | OPERATIONS                 |
|--------------------|-------------|---|----------------------------|
| Basic HTML         | CKEditor    | Authenticated user,Administrator                                | <button>Configure</button> |
| Text only controls | CKEditor    | Anonymous user,Administrator                                    | <button>Configure</button> |
| Full HTML          | CKEditor    | Administrator   | <button>Configure</button> |
| Restricted HTML    | —           | Anonymous user,Administrator                                    | <button>Configure</button> |
| Plain text         | —           | <i>This format is shown when no other formats are available</i> | <button>Configure</button> |

7. That's it! If you visit a page that allows comments from anonymous users, the newly created format will be used to capture those comments:

Comment \*

B U I := :=

body p

Text format  About text formats ⓘ

## What just happened?

You just added a new CKEditor profile that targets anonymous users on your website, giving them the ability to input rich text and lists.

## Classic editor and inline editing

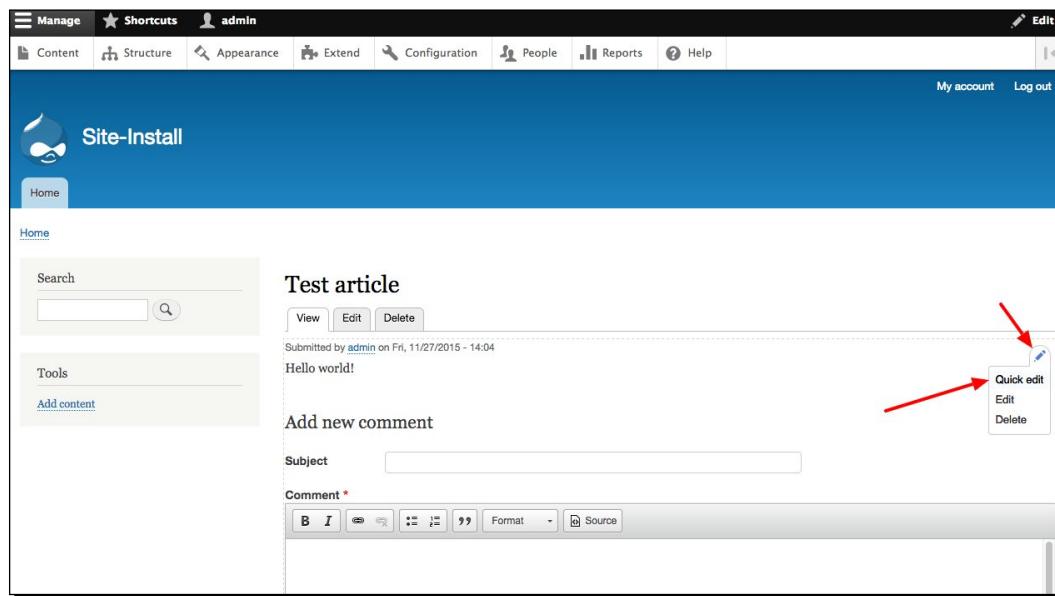
The examples we have seen so far used CKEditor at the backend. This is based on an iframe being added to the page, and they do not share the CSS styles as they appear on the frontend. You can, however, customize the CSS of the editor by including the following in your .info.yml theme file:

```
ckeditor_stylesheets [] = css/ckeditor-iframe.css
```

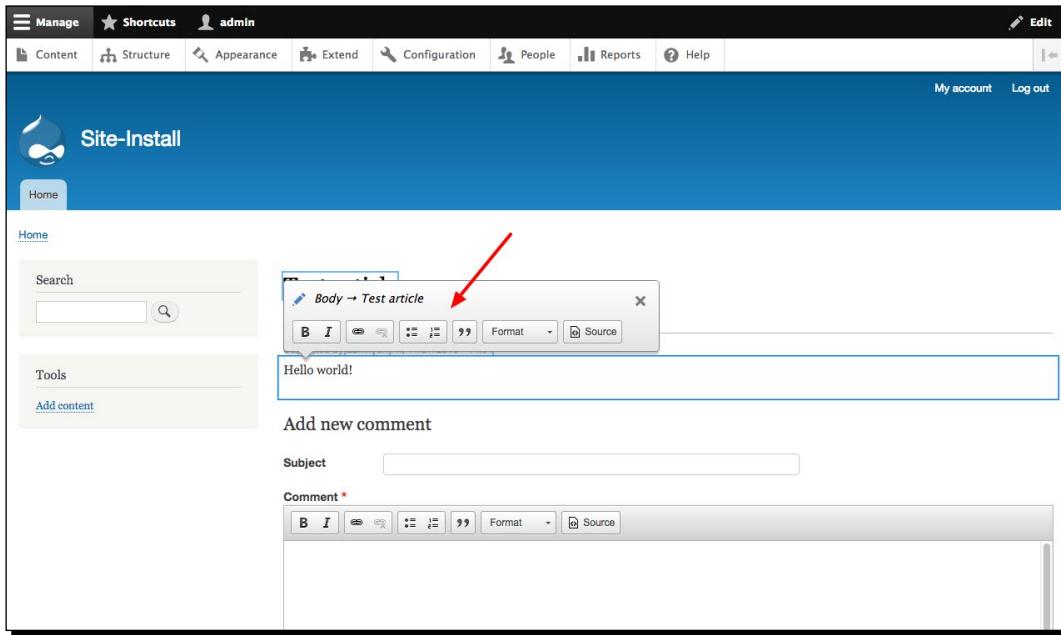
An exciting new possibility in Drupal 8 is the use of inline editing. This is used for quick in-place edits and is used without an iframe. Inline editing gives editors the ability to quickly edit a piece of content without the need to edit the full node. CSS styles are inherited from the theme, leading to a real WYSIWYG experience.

## Time for action – using inline editing

1. Ensure that the **Quick edit** module is enabled.
2. Click on the pencil icon in the top left of the page and select **Quick edit**:



- 3.** The new editor instance opens at the top of the content area, and you can start editing text straightaway:



- 4.** That's it! You can now use this instead of visiting the edit form to quickly edit things on the fly.

## Adding widgets to CKEditor

CKEditor comes preconfigured with a series of buttons that can be added to a profile. As a programmer, you can extend CKEditor and add your own buttons. This can happen by adding plugins or widgets. The difference between the two is that widgets are plugins that group the behavior of more than one component. An example of a widget is an image where the image itself, alternative text, and a caption form an item and they can be moved around the WYSIWYG area as one item.

To use additional widgets, you will need to do two things:

1. Download or create a plugin for CKEditor.
2. Tell the Drupal core that a new CKEditor plugin should be loaded.

To learn more about the CKEditor side of things, read the documentation on adding CKEditor plugins or CKEditor widgets. Plugins and widgets for CKEditor can be downloaded from <http://ckeditor.com/addons/plugins/all>.

Once you have a CKEditor plugin, you need to tell Drupal core that a new CKEditor plugin needs to be loaded via `\Drupal\ckeditor\CKEditorPluginInterface`. This will create a 1:1 relationship between the CKEditor JavaScript plugin and the Drupal CKEditor Plugin plugin (a confusing name, nonetheless). A default implementation is provided via `Drupal\ckeditor\CKEditorPluginBase`, so not every method needs to be implemented by CKEditor plugins.

Additionally, the following interfaces can be implemented on the Drupal side to extend the plugin functionality:

- ◆ `\Drupal\ckeditor\CKEditorPluginButtonsInterface` allows a CKEditor plugin to define which buttons it provides so that users can configure a CKEditor toolbar instance via the drag-and-drop-based UI
- ◆ `\Drupal\ckeditor\CKEditorPluginContextualInterface` allows a CKEditor plugin to enable itself automatically based on the context: if some other CKEditor plugin's button is enabled, if some filter is enabled, if some CKEditor plugin's setting has a certain value, or a combination of all of these
- ◆ `\Drupal\ckeditor\CKEditorPluginConfigurableInterface` allows a CKEditor plugin to define a settings form to configure any settings that this CKEditor plugin may have
- ◆ `\Drupal\ckeditor\CKEditorPluginCssInterface` allows a CKEditor plugin to define additional CSS to be loaded in iframe instances of CKEditor

To do that, you will need to create a new module utilizing the Drupal Plugin API. Even though CKEditor and Drupal share the notation of plugins, note that these are very different things. As per the Drupal Plugin API, plugin implementations must be annotated with the `@CKEditorPlugin` annotation so that they can be discovered.

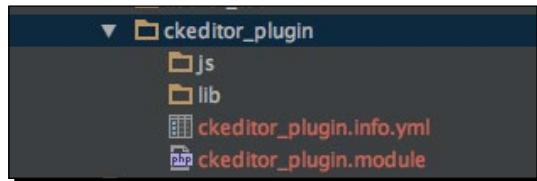
When creating CKEditor plugins, keep in mind that you are creating functionality intended for content editors, so the user interface and user experience should be excellent. Have a look at `ckeditor.drupalimage.admin.js` and `ckeditor.stylescombo.admin.js` for examples of good implementations (see also <https://www.drupal.org/node/2567801>).

We will extensively make use of the Drupal Plugin API in the next section of this chapter when adding a new block, but Drupal plugins are essentially swappable pieces of functionality.

## Have a go hero - create a CKEditor plugin and allow Drupal to discover it

To declare a Drupal instance of a CKEditor plugin, you need to follow these steps:

1. Create your module structure as follows. Within `js`, the CKEditor plugin is placed, and the Drupal plugin is placed within `lib`:



2. Ensure that the CKEditor plugin (named `plugin.js` within the `js` directory) is namespaced as Drupal expects it:

```
(function ($, Drupal, drupalSettings, CKEDITOR) {
  /* existing plugin code */
}) (jQuery, Drupal, drupalSettings, CKEDITOR)
```

3. Create the file that will contain the class:

```
DRUPAL_ROOT/modules/MODULE_NAME/lib/Drupal/PLUGIN_NAME/Plugin
/CKEditorPlugin/PLUGIN_NAME.php
```

4. Extend the `CKeditorPluginBase` class within that file:

```
<?php
/**
 * @file
 * Contains \Drupal\PLUGIN_NAME\Plugin\CKEditorPlugin\PLUGIN_NAME.
 */

namespace Drupal\PLUGIN_NAME\Plugin\CKEditorPlugin;

use Drupal\ckeditor\CKEditorPluginBase;
use Drupal\Core\Plugin\PluginBase;
use Drupal\ckeditor\CKEditorPluginInterface;
use Drupal\ckeditor\CKEditorPluginButtonsInterface;
use Drupal\ckeditor\CKEditorPluginConfigurableInterface;
use Drupal\ckeditor\CKEditorPluginContextualInterface;
use Drupal\editor\Entity\Editor;
```

```
use Drupal\ckeditor\Annotation\CKEditorPlugin;
use Drupal\Core\Annotation\Translation;

/**
 * Defines the "PLUGIN_NAME" plugin.
 * @see MetaContextual
 * @see MetaButton
 * @see MetaContextualAndButton
 *
 * @CKEditorPlugin(
 *   id = "PLUGIN_NAME",
 *   label = @Translation("PLUGIN_NAME")
 * )
 */

class PLUGIN_NAME extends CKEditorPluginBase
implements CKEditorPluginConfigurableInterface,
CKEditorPluginContextualInterface {

  // your code here

}
```

For further information, you can visit the documentation for the `Drupal\ckeditor` namespace at <https://api.drupal.org/api/drupal/namespace/Drupal%21ckeditor/8.2.x>.

## Introduction to the Block API for Drupal 8

In Drupal 7, adding a block was just a matter of implementing a hook such as `hook_block_info()` and a few more in a custom module. The block then was available in the user interface for you to place wherever you wanted.

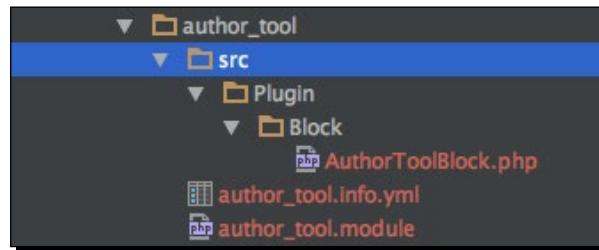
In Drupal 8, custom blocks provided by a module implement the Block Plugin API, which is a subset of the more generic Plugin API. What used to be info hooks that returned arrays for block discovery is now composed of Annotations and the use of PSR-0, so Drupal can both find and understand your blocks. Callback functions that returned the content of your block are now methods on `Drupal\block\BlockPluginInterface` that we can override as needed in our custom block code.

Creating a block in Drupal 8 requires creating a plugin according to the Plugin API and annotation-based plugin discovery. Throughout the book, we have seen both of these at play when creating field widgets or CKEditor profiles.

A workflow of creating a block can be visualized as follows:

1. Create a block plugin using annotations.
2. Implement the `Drupal\Core\Block\BlockBase` class.
3. Implement the `Drupal\Core\Block\BlockPluginInterface` class methods according to the use case.

For a custom block named `author_tool`, a PSR-4-compliant structure is `author_tool/src/Plugin/Block`:



## Time for action – creating a block to aid the authoring experience

Let's create a custom block that will be available to authenticated users to create new recipes when visiting a recipe page. Our module will be called `author_tool`:

1. Create a structure similar to what we saw in the last screenshot.
2. Create a `author_tool.info.yml` file and create a dependency of your module with the block system:

```

name: Author tool
type: module
description: A custom block to allow content editors to quickly
add a new recipe.
core: 8.x
dependencies:
  - block

```

3. Extend the `BlockBase` class with your own implementation, `AuthorToolBlock`, and place the following code in it:

```

<?php

/**
 * @file

```

```
* Contains \Drupal\author_tool\Plugin\Block\AuthorToolBlock.php.  
*/  
  
namespace Drupal\author_tool\Plugin\Block;  
  
use Drupal\Core\Block\BlockBase;  
  
/**  
 * Provides a custom block.  
 *  
 * Drupal\Core\Block\BlockBase gives us a very useful set of  
 * basic functionality  
 * for this configurable block. We can just fill in a few of  
 * the blanks with  
 * defaultConfiguration(), blockForm(), blockSubmit(), and  
 * build().  
 *  
 * @Block(  
 *   id = "author_tool_block",  
 *   admin_label = @Translation("Author tool block")  
 * )  
 */  
class AuthorToolBlock extends BlockBase {  
  
  // our code goes here.  
  
}
```

- 4.** Within your class, implement the `BlockPluginInterface::build` method like this:

```
/**  
 * {@inheritDoc}  
 */  
public function build() {  
  
}
```

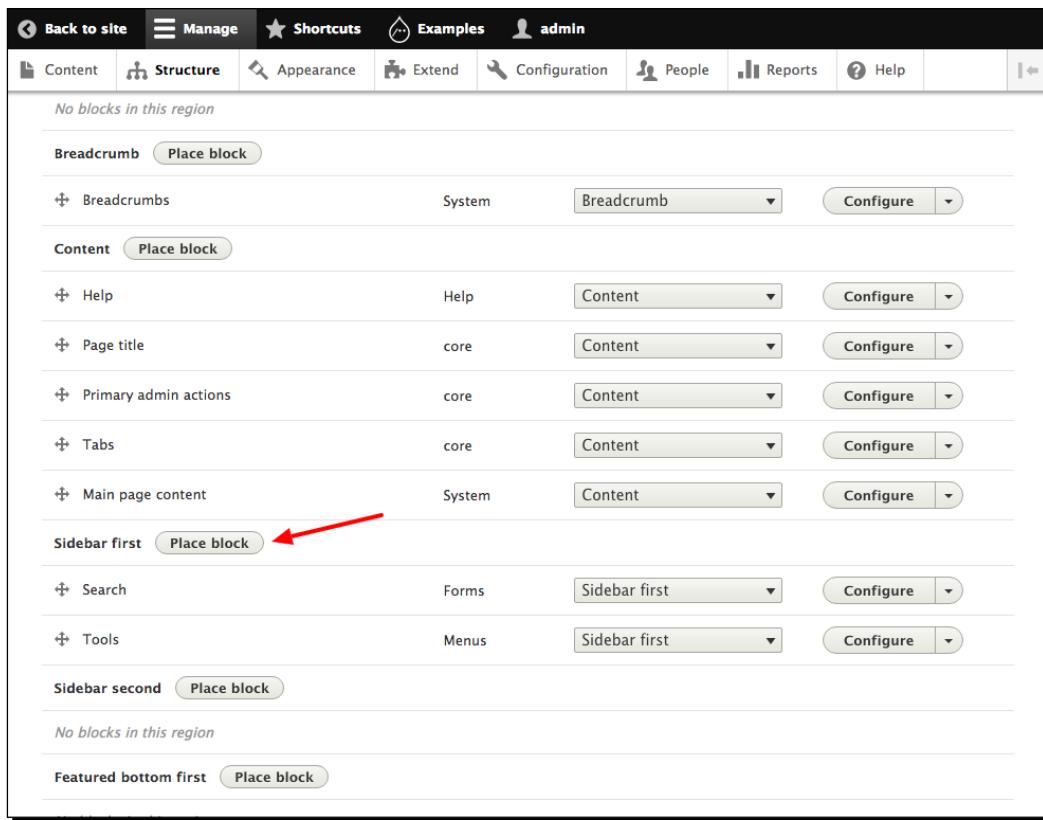
- 5.** The build method is returning a renderable array or content like `hook_block_view` did for Drupal 7. Render arrays are the preferred way in Drupal 8. The complete implementation is shown as follows:

```
<?php  
  
/**  
 * @file
```

```
* Contains \Drupal\author_tool\Plugin\Block\AuthorToolBlock.php.  
*/  
  
namespace Drupal\author_tool\Plugin\Block;  
  
use Drupal\Core\Block\BlockBase;  
  
/**  
 * Provides a custom block.  
 *  
 * Drupal\Core\Block\BlockBase gives us a very useful set of  
 * basic functionality  
 * for this configurable block. We can just fill in a few of  
 * the blanks with  
 * defaultConfiguration(), blockForm(), blockSubmit(), and  
 * build().  
 *  
 * @Block(  
 *   id = "author_tool_block",  
 *   admin_label = @Translation("Author tool block")  
 * )  
 */  
class AuthorToolBlock extends BlockBase {  
  
    /**  
     * {@inheritDoc}  
     */  
    public function build() {  
  
        $link = array(  
            '#type' => 'markup',  
            '#title' => 'Author tools',  
            '#markup' => $this->t('<a href=:url>Add another  
            recipe</a>', array(':url' => 'add/recipe')),  
        );  
  
        return $link;  
    }  
}
```

6. Your module is ready. Go to **Extend**, locate it, and enable it.

7. Now it's time to place your block on the recipe pages for authenticated users to see. From the main menu, select **Structure | Block Layout**, scroll down to **Sidebar first**, and click on the **Place block** button:

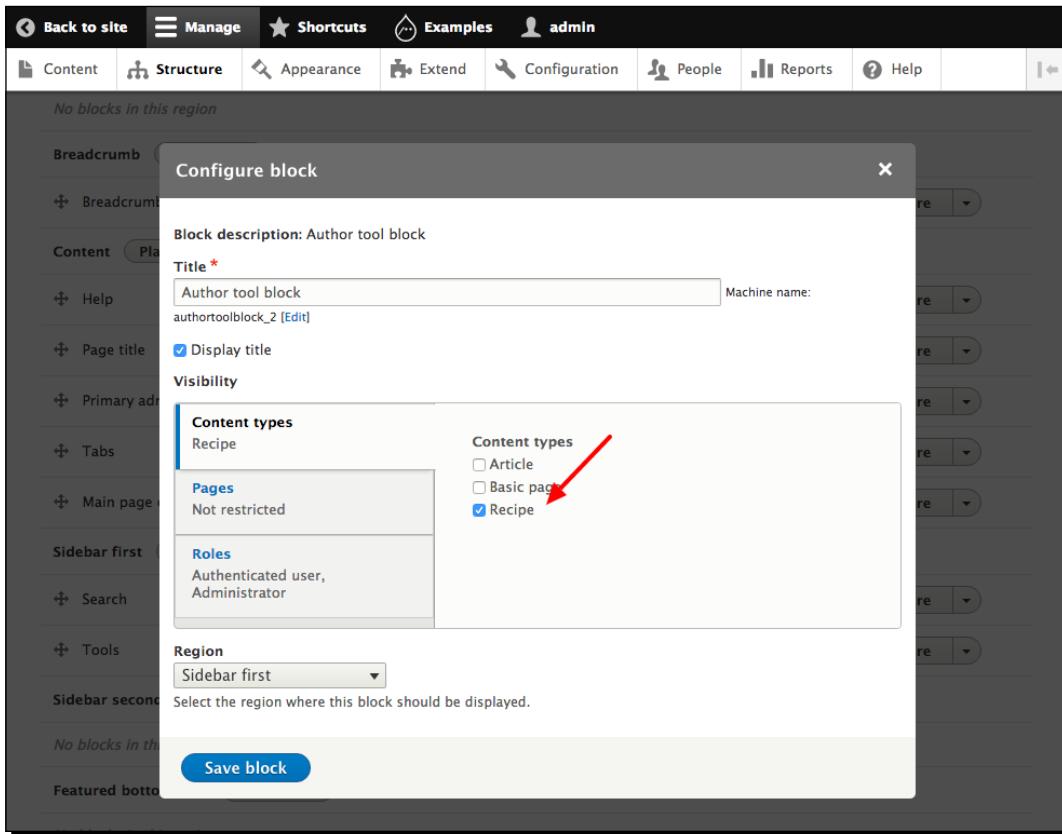


**8.** Locate your block and click on **Place block**:

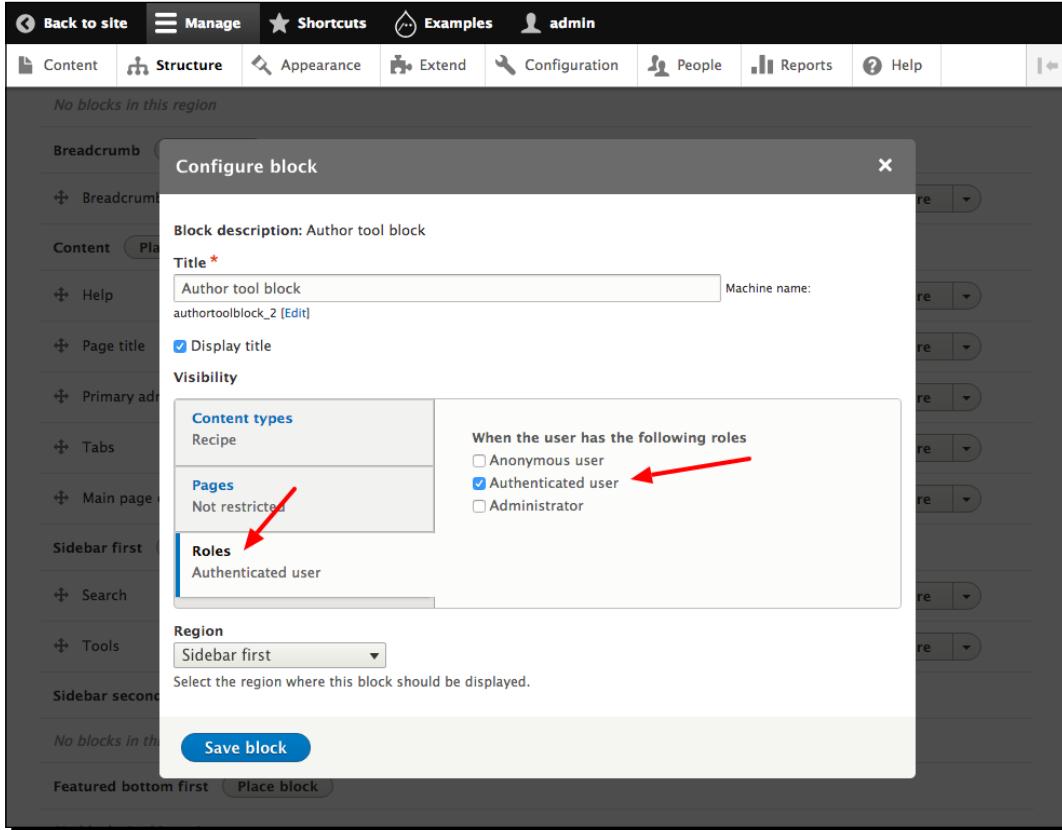
The screenshot shows the 'Place block' modal dialog from the Drupal interface. The dialog has a header 'Place block' with a close button 'X'. On the left is a sidebar with various block categories like 'Breadcrumb', 'Content', 'Help', etc. The main area lists blocks with columns for 'BLOCK', 'CATEGORY', and 'OPERATIONS'. The 'OPERATIONS' column contains a 'Place block' button for each row. A red arrow points to the 'Place block' button for the first item, 'Author tool block'. The table data is as follows:

| BLOCK  | CATEGORY      | OPERATIONS         |
|--|---------------|--------------------|
| Author tool block                                | Author tool   | <b>Place block</b> |
| Example: empty block                             | Block Example | <b>Place block</b> |
| Example: uppercase this please                   | Block Example | <b>Place block</b> |
| Title of first block (example_configurable_text) | Block Example | <b>Place block</b> |
| Page title                                       | core          | <b>Place block</b> |
| Primary admin actions                            | core          | <b>Place block</b> |
| Tabs   | core          | <b>Place block</b> |
| Search form                                      | Forms         | <b>Place block</b> |
| User login                                       | Forms         | <b>Place block</b> |
| Help   | Help          | <b>Place block</b> |

**9.** Configure your block to display only recipe content types:



**10.** Configure your block to display for authenticated users only:



**11.** Save the image, and that's it! Now visit a recipe page to verify that it is there:

The screenshot shows the Drupal 8 Site-Install interface. At the top, there is a navigation bar with links for Manage, Shortcuts, Examples, admin, Content, Structure, Appearance, Extend, Configuration, People, Reports, Help, My account, and Log out. Below the navigation bar, the title "Site-Install" is displayed next to a blue circular logo. A breadcrumb trail shows "Home". The main content area displays a "Recipe sample" node. On the left side of the content area, there is a sidebar with "Author tools" containing a link to "Add another recipe" which has a red arrow pointing to it. Below this is a "Search" section with a search input field and a magnifying glass icon. At the bottom of the sidebar, there is a "Tools" section with links to "Add content" and "Block Example". The main content area shows the node title "Recipe sample", submission details ("Submitted by admin on Tue, 12/08/2015 - 08:00"), and three action buttons: View, Edit, and Delete.

## ***What just happened?***

Although this is a very simple example of creating a block through code, the simplicity of creating a block through the newly arrived Block API in Drupal 8 is clear. You created a block to aid with the author UI and learned the basics of extending the Plugin API to add custom functionality to the system.

To explore further, have a look at <https://api.drupal.org/api/drupal> for the `BlockBase` class to get a sense of all the methods there that can be added.

## Time for action – including default configuration in your module

Once you have placed your block in your theme, you may wish to include that configuration when the module installs. By including a default configuration at installation time, you are able to provide a sensible default for placing that block. Also, within your deployment mechanism, all you have to worry about is installing the module, and the rest of the configuration will be applied at that step:

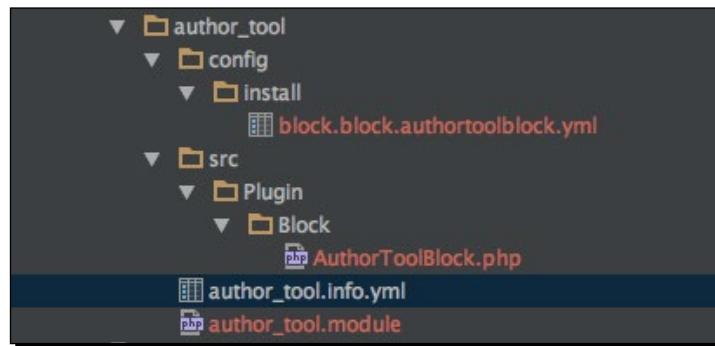
1. Export the configuration by going to **Structure | Configuration | Configuration synchronization | Export | Single item**, and choose **Block** and **Author tool block**.
2. Make sure you copy and paste everything but the `uuid` line (shown in the following screenshot):

The screenshot shows the 'Single export' configuration page in the Drupal admin interface. The 'Configuration type' is set to 'Block'. The 'Configuration name' dropdown is set to 'Author tool block'. A red arrow points to this dropdown. Another red arrow points to the 'uuid' line in the configuration code below.

```

uuid: 08f4f9ce-fcb3-45a0-8b0f-15e381995ce6
langcode: en
status: false
dependencies:
  module:
    - author_tool
    - node
    - user
  theme:
    - bartik
id: authortoolblock
theme: bartik
region: '-1'
weight: -7
provider: null
plugin: author_tool_block
settings:
  id: author_tool_block
  label: 'Author tool block'
  
```

3. Within your module, create the following structure and place the contents in a file named `block.block.authortoolblock.yml`:



4. That's it again! Try uninstalling and installing the module again. The block should be placed automatically at exactly the same position with the same visibility rules.

## Summary

In this chapter, you learned how to enhance the author UI by installing, configuring, and reusing that configuration of the built-in WYSIWYG editor, which is a part of Drupal 8 core now. You explored both the backend editor instance as well as the inline editing capabilities, and had a go of adding your own CKEditor plugin. You also learned how to use the new Block API, add a new block to the system through a custom module, extend default methods, and provide default configuration when installing modules.

In the next chapter, we are going to see how to work with media and integrate them into our new Drupal 8 site.

# 7

## Adding Media to Our Site

*A text-only site is not going to hold the interest of visitors; a site needs some pizzazz and some spice! One way to add some pizzazz to your site is by adding some multimedia content, such as images, video, audio, and so on. But, we don't just want to add a few images here and there; in fact, we want an immersive and compelling multimedia experience that is easy to manage, configure, and extend. The File entity ([https://drupal.org/project/file\\_entity](https://drupal.org/project/file_entity)) module for Drupal 8 will enable us to manage files very easily. In this chapter, we will discover how to integrate the File entity module to add images to our d8dev site, and will explore compelling ways to present images to users. This will include taking a look at the integration of a lightbox-type UI element for displaying the File-entity-module-managed images, and learning how we can create custom image styles through UI and code.*

The following topics will be covered in this chapter:

- ◆ The File entity module for Drupal 8
- ◆ Adding a Recipe image field to your content types
- ◆ Code example—image styles for Drupal 8
- ◆ Displaying recipe images in a lightbox popup
- ◆ Working with Drupal issue queues

## **Introduction to the File entity module**

As per the module page at [https://www.drupal.org/project/file\\_entity](https://www.drupal.org/project/file_entity):

*File entity provides interfaces for managing files. It also extends the core file entity, allowing files to be fieldable, grouped into types, viewed (using display modes) and formatted using field formatters. File entity integrates with a number of modules, exposing files to Views, Entity API, Token and more.*

In our case, we need this module to easily edit image properties such as Title text and Alt text. So these properties will be used in the colorbox popup to display them as captions.

## **Working with dev versions of modules**

There are times when you come across a module that introduces some major new features and is fairly stable, but not quite ready for use on a live/production website, and is therefore available only as a dev version. This is a perfect opportunity to provide a valuable contribution to the Drupal community. Just by installing and using a dev version of a module (in your local development environment, of course), you are providing valuable testing for the module maintainers. Of course, you should enter an issue in the project's issue queue if you discover any bugs or would like to request any additional features. Also, using a dev version of a module presents you with the opportunity to take on some custom Drupal development. However, it is important that you remember that a module is released as a dev version for a reason, and it is most likely not stable enough to be deployed on a public-facing site.

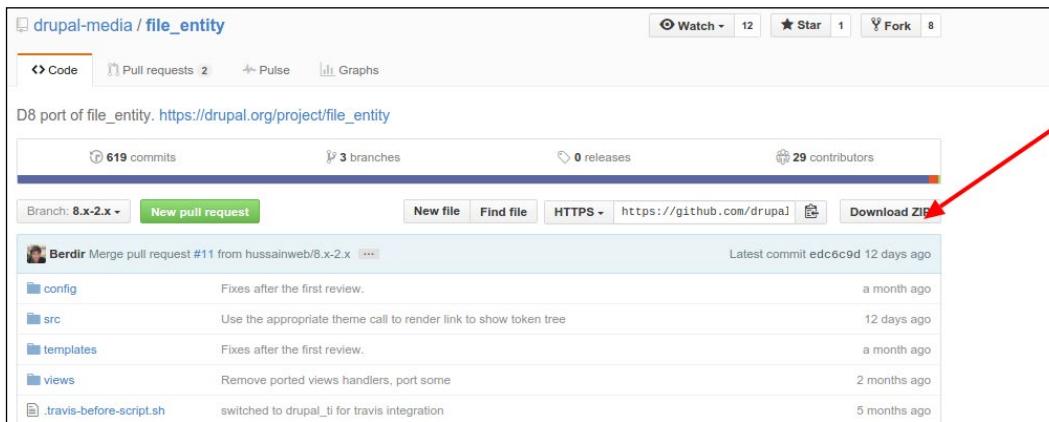
Our use of the File entity module in this chapter is a good example of working with the dev version of a module. One thing to note: Drush will download official and dev module releases. But at this point in time, there is no official port for the File entity module in Drupal, so we will use the unofficial one, which lives on GitHub ([https://github.com/drupal-media/file\\_entity](https://github.com/drupal-media/file_entity)). In the next step, we will be downloading the dev release with GitHub.

### **Time for action – installing a dev version of the File entity module**

In Drupal, we use Drush to download and enable any module/theme, but there is no official port yet for the file entity module in Drupal, so we can use the unofficial one, which lives on GitHub at [https://github.com/drupal-media/file\\_entity](https://github.com/drupal-media/file_entity):

1. Open the Terminal (Mac OS X) or Command Prompt (Windows) application, and go to the root directory of your d8dev site.

- 2.** Go inside the `modules` folder and download the File entity module from GitHub. We use the `git` command to download this module: `$ git clone https://github.com/drupal-media/file_entity`. Another way is to download a `.zip` file from [https://github.com/drupal-media/file\\_entity](https://github.com/drupal-media/file_entity) and extract it in the `modules` folder:



- 3.** Next, on the **Extend** page (`admin/modules`), enable the File entity module.

### **What just happened?**

We enabled the File entity module, and learned how to download and install with GitHub.

## A new recipe for our site

In this chapter, we are going to create a new recipe: Thai Basil Chicken. If you would like to have more real content to use as an example, and feel free to try the recipe out!



- ◆ **Name:** Thai Basil Chicken
- ◆ **Description:** A spicy, flavorful version of one of my favorite Thai dishes
- ◆ **RecipeYield :** Four servings
- ◆ **PrepTime:** 25 minutes
- ◆ **CookTime:** 20 minutes
- ◆ **Ingredients :**
  - One pound boneless chicken breasts
  - Two tablespoons of olive oil
  - Four garlic cloves, minced
  - Three tablespoons of soy sauce
  - Two tablespoons of fish sauce
  - Two large sweet onions, sliced
  - Five cloves of garlic
  - One yellow bell pepper
  - One green bell pepper
  - Four to eight Thai peppers (depending on the level of hotness you want)
  - One-third cup of dark brown sugar dissolved in one cup of hot water
  - One cup of fresh basil leaves
  - Two cups of Jasmin rice
- ◆ **Instructions:**
  - Prepare the Jasmine rice according to the directions.
  - Heat the olive oil in a large frying pan over medium heat for two minutes.
  - Add the chicken to the pan and then pour on soy sauce.
  - Cook the chicken until there is no visible pinkness—approximately 8 to 10 minutes.
  - Reduce heat to medium low.
  - Add the garlic and fish sauce, and simmer for 3 minutes.
  - Next, add the Thai chilies, onion, and bell pepper and stir to combine.
  - Simmer for 2 minutes.
  - Add the brown sugar and water mixture. Stir to mix, and then cover.
  - Simmer for 5 minutes.
  - Uncover, add basil, and stir to combine.
  - Serve over rice.

## Time for action – adding a Recipe images field to our Recipe content type

We will use the manage fields administrative page to add a Media field to our d8dev Recipe content type:

1. Open up the d8dev site in your favorite browser, click on the **Structure** link in the **Admin** toolbar, and then click on the **Content types** link.
2. Next, on the **Content types** administrative page, click on the **Manage fields** link for your Recipe content type:

The screenshot shows the 'Content types' administrative page. It lists three content types: Article, Basic page, and Recipe. For each content type, there is a 'DESCRIPTION' column and an 'OPERATIONS' column containing a 'Manage fields' button. A red arrow points to the 'Manage fields' button for the 'Recipe' content type.

3. Now, on the **Manage fields** administrative page, click on the **Add field** link. On the next screen, select **Image** from the **Add a new field** dropdown and **Label as Recipe images**. Click on the **Save field settings** button.
4. Next, on the **Field settings** page, select **Unlimited** as the allowed number of values. Click on the **Save field settings** button. On the next screen, leave all settings as they are and click on the **Save settings** button.
5. Next, on the **Manage form display** page, select widget **Editable file** for the **Recipe images** field and click on the **Save** button.
6. Now, on the **Manage display** page, for the **Recipe images** field, select **Hidden** as the label. Click on the settings icon. Then select **Medium (220\*220)** as the image style, and click on the **Update** button. At the bottom, click on the **Save** button:

The screenshot shows the 'Manage display' page for the 'Recipe images' field. It has two tabs: 'Label' and 'Field settings'. The 'Label' tab is selected, showing a dropdown menu with 'Hidden' selected. The 'Field settings' tab is also visible. Red arrows point to the 'Hidden' dropdown and the 'Medium (220\*220)' dropdown under 'Image style'. At the bottom right, there are 'Update' and 'Cancel' buttons.

7. Let's add some Recipe images to a recipe. Click on the **Content** link in the menu bar, and then click on **Add content and Recipe**. On the next screen, fill in the title as **Thai Basil Chicken** and other fields respectively as mentioned in the preceding recipe details.
8. Now, scroll down to the new **Recipe images** field that you have added. Click on the **Add a new file** button or drag and drop images that you want to upload. Then click on the **Save and Publish** button:

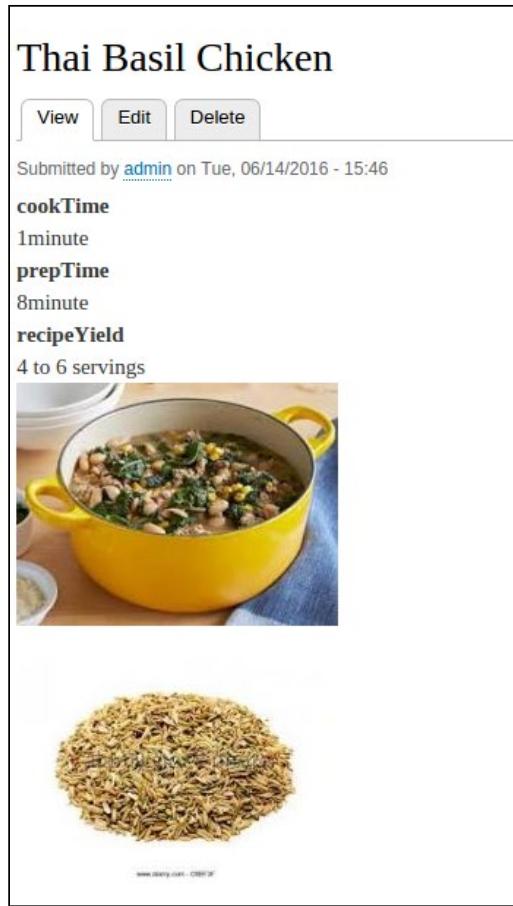
The screenshot shows a 'FILE INFORMATION' table with six rows of uploaded files, each with 'Edit' and 'Remove' buttons. Below the table is a section for adding new files with a 'Choose Files' input field set to 'No file chosen'. A note specifies an unlimited number of files can be uploaded with a 2 MB limit and allowed types (png, gif, jpg, jpeg). At the bottom are 'Save and publish' and 'Preview' buttons.

| FILE INFORMATION                             | OPERATIONS                                    |
|--|---|
| ⊕  tablespoon_of_ground_cumin_medium.jpg     | <button>Edit</button> <button>Remove</button> |
| ⊕  Tablespoon-of-Fennel-Seed-Stock-Photo.jpg | <button>Edit</button> <button>Remove</button> |
| ⊕  salt-on-a-teaspoon.jpg                    | <button>Edit</button> <button>Remove</button> |
| ⊕  onion-chopped.jpg                         | <button>Edit</button> <button>Remove</button> |
| ⊕  olive-oil.jpg                             | <button>Edit</button> <button>Remove</button> |
| ⊕  mincing-garlic.jpg                        | <button>Edit</button> <button>Remove</button> |

**Add a new file**  
 No file chosen  
Unlimited number of files can be uploaded to this field.  
2 MB limit.  
Allowed types: png gif jpg jpeg.

**Save and publish**  **Preview**

9. Reload your **Thai Basil Chicken** recipe page, and you should see something similar to the following:



- 10.** All the images are stacked on top of each other. So, we will add the following CSS just under the style for `field--name-field-recipe-images` and `field--type-recipe-images` in the `/modules/d8dev/styles/d8dev.css` file, to lay out the Recipe images in more of a grid:

```
.node .field--type-recipe-images {  
    float: none !important;  
}  
.field--name-field-recipe-images .field__item {  
    display: inline-flex;  
    padding: 6px;  
}
```

**11.** Now we will load this `d8dev.css` file to affect this grid style. In Drupal 8, loading a CSS file has a process:

1. Save the CSS to a file.
2. Define a library, which can contain the CSS file.
3. Attach the library to a render array in a hook.

**12.** So, we have already saved a CSS file called `d8dev.css` under the `styles` folder; now we will define a library. To define one or more (asset) libraries, add a `*.libraries.yml` file to your theme folder. Our module is named `d8dev`, and then the filename should be `d8dev.libraries.yml`. Each library in the file is an entry detailing CSS, like this:

```
d8dev:  
  version: 1.x  
  css:  
    theme:  
      styles/d8dev.css: {}
```

**13.** Now, we define the `hook_page_attachments()` function to load the CSS file. Add the following code inside the `d8dev.module` file. Use this hook when you want to conditionally add attachments to a page:

```
/**  
 * Implements hook_page_attachments().  
 */  
function d8dev_page_attachments(array &$attachments) {  
  $attachments['#attached']['library'][] = 'd8dev/d8dev'; }
```

**14.** Now, we will need to clear the cache for our `d8dev` site by going to **Configuration**, clicking on the **Performance** link, and then clicking on the **Clear all caches** button. Reload your **Thai Basil Chicken** recipe page, and you should see something similar to the following:

## Thai Basil Chicken

[View](#) [Edit](#) [Delete](#)

Submitted by [\[REDACTED\]](#) on Tue, 06/14/2016 - 15:46

**cookTime**

1 minute

**prepTime**

8 minute

**recipeYield**

4 to 6 servings

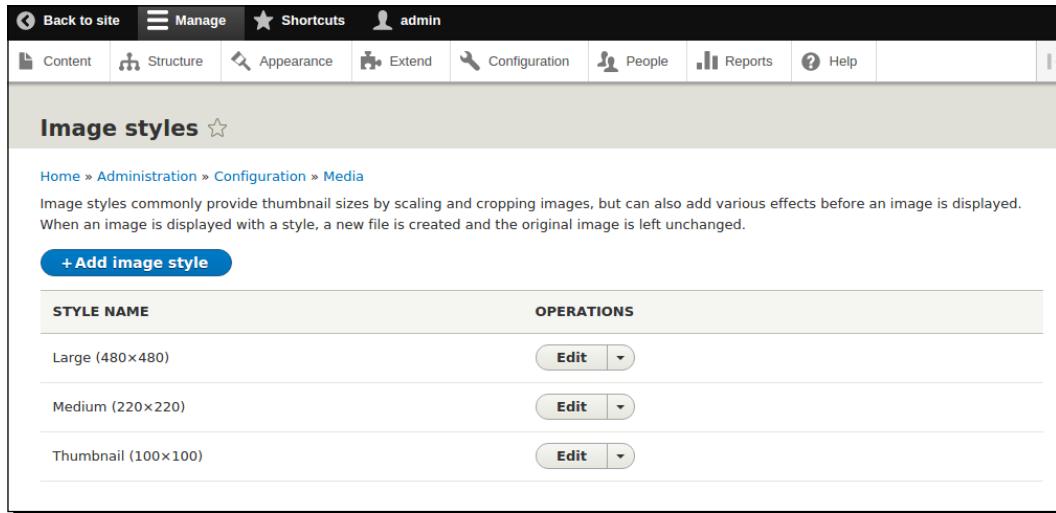


### **What just happened?**

We added and configured a media-based field for our Recipe content type. We updated the d8dev module with custom CSS code to lay out the Recipe images in more of a grid format. And also we looked at how to attach a CSS file through a module.

## Creating a custom image style

Before we configure a colorbox feature, we are going to create a custom image style to use when we add them in colorbox content preview settings. Image styles for Drupal 8 are part of the core Image module. The core image module provides three default image styles—thumbnail, medium, and large—as seen in the following Image style configuration page:



The screenshot shows the 'Image styles' configuration page in the Drupal 8 admin interface. The top navigation bar includes links for Back to site, Manage, Shortcuts, and admin. Below the navigation is a toolbar with links for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. The main content area is titled 'Image styles' with a star icon. It displays a breadcrumb trail: Home > Administration > Configuration > Media. A note below the title states: 'Image styles commonly provide thumbnail sizes by scaling and cropping images, but can also add various effects before an image is displayed. When an image is displayed with a style, a new file is created and the original image is left unchanged.' A blue 'Add image style' button is visible. The table lists three styles:

| STYLE NAME          | OPERATIONS                               |
|---------------------|--|
| Large (480x480)     | <button>Edit</button> <button>⋮</button> |
| Medium (220x220)    | <button>Edit</button> <button>⋮</button> |
| Thumbnail (100x100) | <button>Edit</button> <button>⋮</button> |

Now, we are going to add a fifth custom image style, an image style that will resize our images somewhere between the 100 x 75 thumbnail style and the 220 x 165 medium style. We will walkthrough the process of creating an image style through the Image style administrative page, and also walkthrough the process of programmatically creating an image style.

### Time for action – adding a custom image style through the image style administrative page

First, we will use the Image style administrative page (`admin/config/media/image-styles`) to create a custom image style:

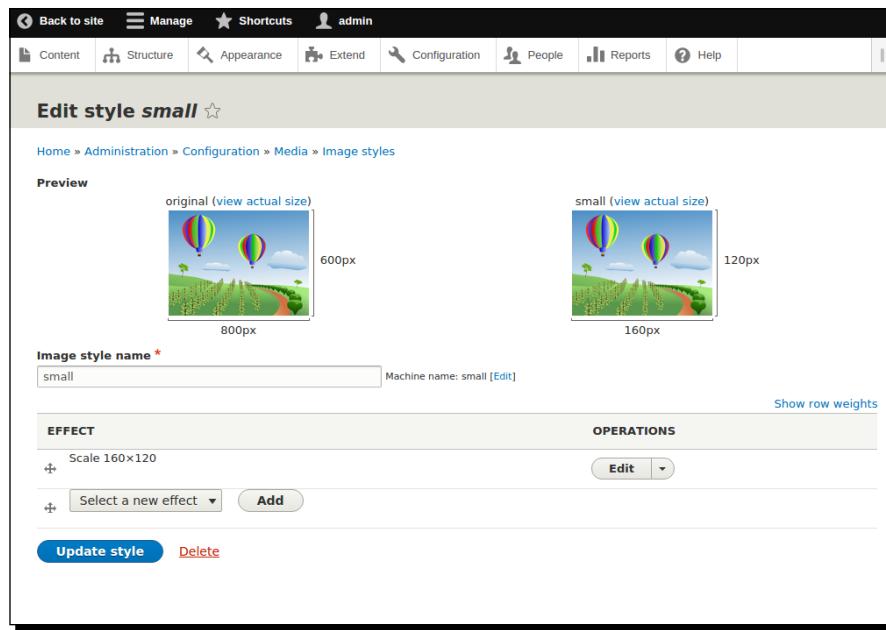
1. Open the d8dev site in your favorite browser, click on the **Configuration** link in the **Admin** toolbar, and click on the **Image** styles link under the **Media** section.
2. Once the Image styles administrative page has loaded, click on the **Add style** link.

- 3.** Next, enter `small` for the **Image style name** of your custom image style, and click on the **Create new style** button:

Now, we will add the one and only effect for our custom image style by selecting **Scale** from the **EFFECT** options and then clicking on the **Add** button.

- 4.** On the **Add Scale effect** page, enter `160` for the width and `120` for the height. Leave the **Allow Upscaling** checkbox unchecked, and click on the **Add effect** button:

5. Finally, just click on the **Update style** button on the **Edit small style** administrative page, and we are done. We now have a new custom small image style that we will be able to use to resize images for our site:



## **What just happened?**

We learned how easy it is to add a custom image style with the administrative UI. Now, we are going to see how to add a custom image style by writing some code. The advantage of having code-based custom image styles is that it will allow us to utilize a source code repository, such as Git, to manage and deploy our custom image styles between different environments. For example, it would allow us to use Git to promote image styles from our development environment to a live production website. Otherwise, the manual configuration that we just did would have to be repeated for every environment.

## **Time for action – creating a programmatic custom image style**

Now, we will see how we can add a custom image style with code:

1. The first thing we need to do is delete the small image style that we just created. So, open your d8dev site in your favorite browser, click on the **Configuration** link in the **Admin** toolbar, and then click on the **Image styles** link under the **Media** section.
2. Once the **Image styles** administrative page has loaded, click on the **delete** link for the small image style that we just added.

- 3.** Next, on the **Optionally select a style before deleting small** page, leave the default value for the **Replacement style** select list as **No replacement, just delete**, and click on the **Delete** button:

- 4.** In Drupal 8, image styles have been converted from an array to an object that extends ConfigEntity. All image styles provided by modules need to be defined as YAML configuration files in the config/install folder of each module.
- 5.** Suppose our module is located at `modules/d8dev`. Create a file called `modules/d8dev/config/install/image.style.small.yml` with the following content:

```
uuid: b97a0bd7-4833-4d4a-ae05-5d4da0503041
langcode: en
status: true
dependencies: { }
name: small
label: small
effects:
  c76016aa-3c8b-495a-9e31-4923f1e4be54:
    uuid: c76016aa-3c8b-495a-9e31-4923f1e4be54
    id: image_scale
    weight: 1
    data:
      width: 160
      height: 120
      upscale: false
```

[  We need to use a UUID generator to assign unique IDs to image style effects. *Do not copy/paste UUIDs* from other pieces of code or from other image styles! ]

6. The name of our custom style is "small". Here, the name and the label are same. For each effect that we want to add to our image style, we will specify the effect itself in the place of key followed by the value, which acts as the settings for the effect. In the case of the **image\_scale** effect that we are using here, we pass in the **width**, **height**, and **upscale** settings. Finally, the value for the **weight** key allows us to specify the order the effects should be processed in, and although it is not very useful when there is only one effect, it becomes important when there are multiple effects.
7. Now, we will need to uninstall and install our d8dev module by going to the **Extend** page. On the next screen click on the **Uninstall** tab, check the **d8dev** checkbox and click on the **Uninstall** button. Now, click on the **List** tab, check **d8dev**, and click on the **Install** button. Then, go back to the **Image styles** administrative page and you will see our programmatically created small image style.

### **What just happened?**

We created a custom image style with some custom code. We then configured our Recipe content type to use our custom image style for images added to the Recipe images field.

## **Integrating the Colorbox and File entity modules**

The File entity module provides interfaces for managing files. For images, we will be able to edit Title text, Alt text, and Filenames easily. However, the images are taking up quite a bit of room. Let's create a pop-up lightbox gallery and show images in a popup. When someone clicks on an image, a lightbox will pop up and allow the user to cycle through larger versions of all associated images.

### **Time for action – installing the Colorbox module**

Before we can display Recipe images in a Colorbox, we need to download and enable the module:

1. Open the Mac OS X Terminal or Windows Command Prompt, and change to the **d8dev** directory.
2. Next, use Drush to download and enable the current dev release of the Colorbox module (<http://drupal.org/project/colorbox>):

```
$ drush dl colorbox-8.x-1.x-dev
```

```
Project colorbox (8.x-1.x-dev) downloaded to /var/www/html/d8dev/
modules/colorbox.
```

```
[success]
```

```
$ drush en colorbox
```

The following extensions will be enabled: colorbox  
Do you really want to continue? (y/n): y

colorbox was enabled successfully.

```
[ok]
```

3. The Colorbox module depends on the Colorbox jQuery plugin available at <https://github.com/jackmoore/colorbox>. The Colorbox module includes a Drush task that will download the required jQuery plugin at the /libraries directory:

```
$ drush colorbox-plugin  
Colorbox plugin has been installed in libraries
```

```
[success]
```

4. Next, we will look into the Colorbox display formatter. Click on the **Structure** link in the **Admin** toolbar, then click on the **Content types** link, and finally click on the **manage** display link for your Recipe content type under the **Operations** dropdown:

| NAME       | DESCRIPTION   | OPERATIONS                    |
|------------|---|-------------------------------|
| Article    | Use <i>articles</i> for time-sensitive content like news, press releases or blog posts.   | <a href="#">Manage fields</a> |
| Basic page | Use <i>basic pages</i> for your static content, such as an 'About us' page.   | <a href="#">Manage fields</a> |
| Recipe     | A simple recipe content type based on schema.org base HTML5 Microdata schema for Recipes at <a href="https://schema.org/Recipe">https://schema.org/Recipe</a> . | <a href="#">Manage fields</a> |

5. Next, click on the **FORMAT** select list for the **Recipe images** field, and you will see an option for Colorbox, Select as **Colorbox** then you will see the settings change. Then, click on the settings icon:

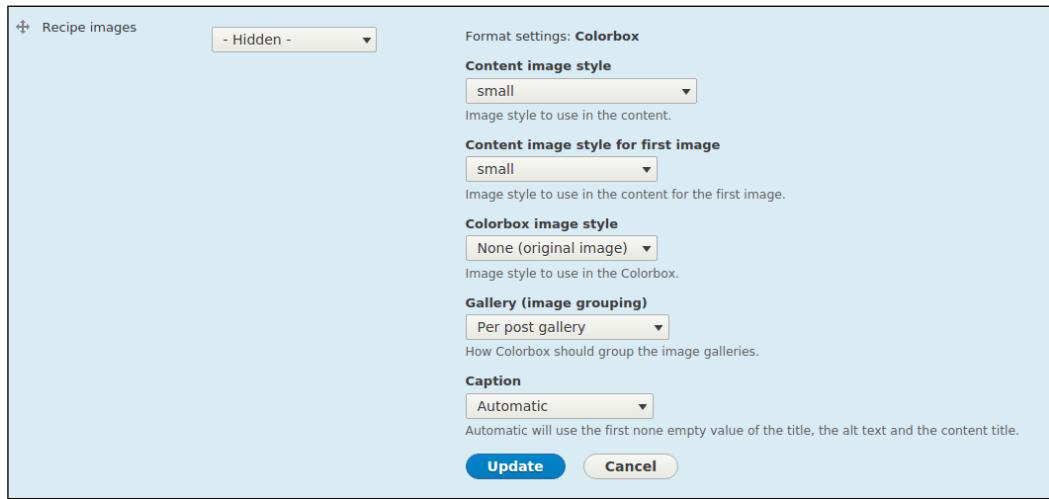
|               |            |          |  |
|---------------|------------|----------|--|
| Body          | - Hidden - | Default  |  |
| Recipe images | - Hidden - | Colorbox |  |

Content image style: Original image  
Colorbox image style: Original image  
Colorbox gallery type: Per post gallery  
Colorbox Caption: Automatic

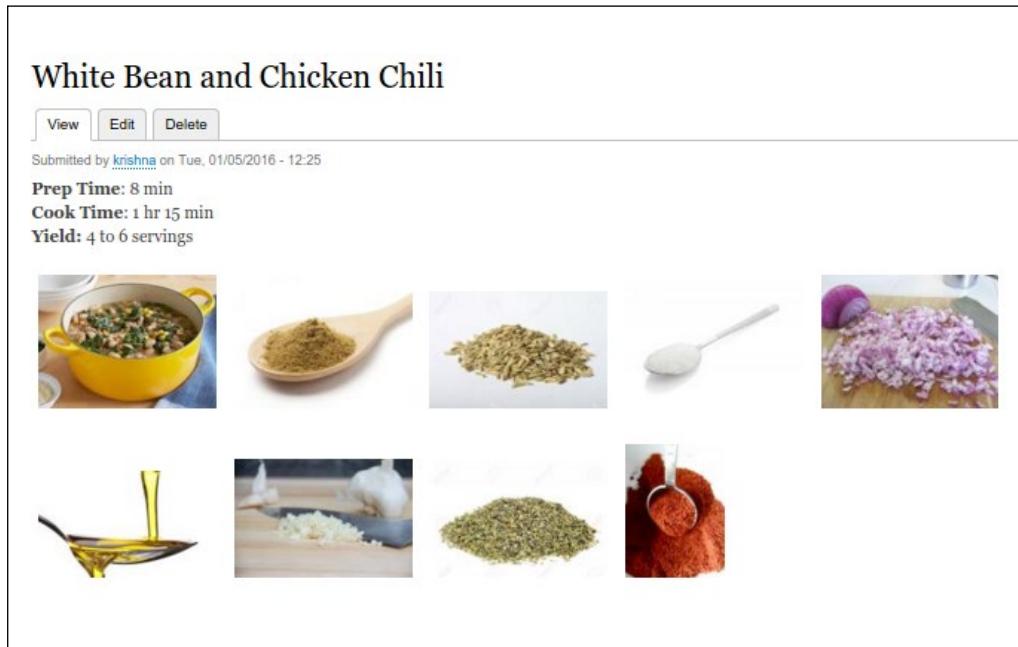
## *Adding Media to Our Site*

---

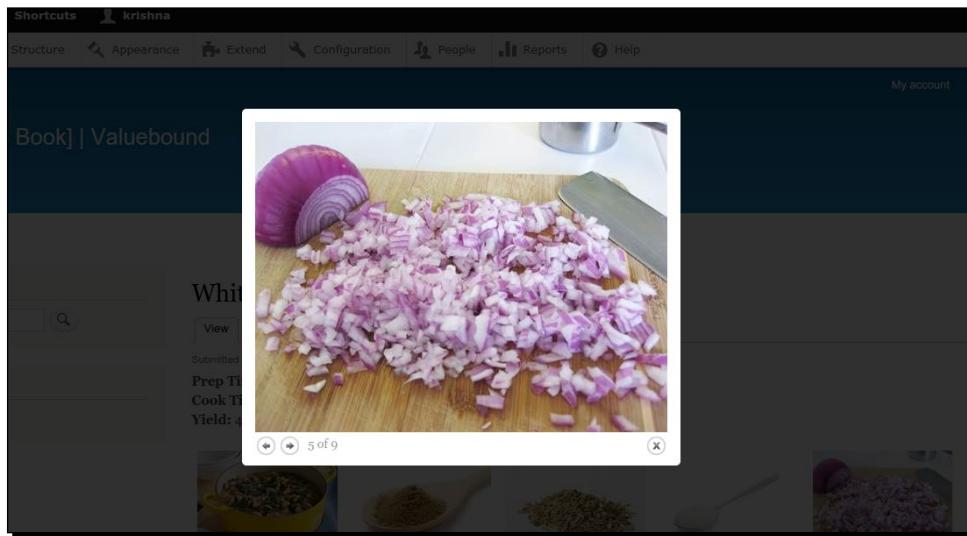
- 6.** Now, you will see the settings for Colorbox. Select **Content image style** as **small** and **Content image style for first image** as **small** in the dropdown, and use the default settings for other options. Click on the **Update** button and next on the **Save** button at the bottom:



- 7.** Reload our **Thai Basil Chicken** recipe page, and you should see something similar to the following (with the new image style, small):



- 8.** Now, click on any image and then you will see the image loaded in the colorbox popup:



- 9.** We have learned more about images for colorbox, but colorbox also supports videos. Another way to add some spice to our site is by adding videos. So there are several modules available to work with colorbox for videos. The Video Embed Field module creates a simple field type that allows you to embed videos from YouTube and Vimeo and show their thumbnail previews simply by entering the video's URL. So you can try this module to add some pizzazz to your site!

### **What just happened?**

We installed the Colorbox module and enabled it for the Recipe images field on our custom Recipe content type. Now, we can easily add images to our d8dev content with the Colorbox pop-up feature.

## **Working with Drupal issue queues**

Drupal has its own issue queue for working with a team of developers around the world. If you need help for a specific project, core, module, or a theme related, you should go to the issue queue, where the maintainers, users, and followers of the module/theme communicate.

The issue page provides a filter option, where you can search for specific issues based on Project, Assigned, Submitted by, Followers, Status, Priority, Category, and so on.

We can find issues at <https://www.drupal.org/project/issues/colorbox>. Here, replace colorbox with the specific module name. For more information, see <https://www.drupal.org/issue-queue>.

In our case, we have one issue with the colorbox module. Captions are working for the **Automatic** and **Content title** properties, but are not working for the **Alt text** and **Title text** properties. To check this issue, go to **Structure | Content types** and click on **Manage display**. On the next screen, click on the settings icon for the Recipe images field. Now select the **Caption** option as **Title text** or **Alt text** and click on the **Update** button. Finally, click on the **Save** button. Reload the **Thai Basil Chicken** recipe page, and click on any image. Then it opens in popup, but we cannot see captions for this.



Make sure you have the Title text and Alt text properties updated for Recipe images field for the Thai Basil Chicken recipe.

## Time for action – creating an issue for the Colorbox module

Now, before we go and try to figure out how to fix this functionality for the Colorbox module, let's create an issue:

1. On <https://www.drupal.org/project/issues/colorbox>, click on the **Create a new issue** link:

The screenshot shows the 'Issues for Colorbox' page on the Drupal 'Download & Extend' site. A red arrow points to the 'Create a new issue' link at the top left of the search form. The search form includes fields for 'Search for', 'Status' (set to 'Open issues'), 'Priority' (set to 'Any'), 'Category' (set to 'Any'), 'Version' (set to 'Any'), and 'Component' (set to 'Any'). Below the search form is a table listing two issues:

| Summary  | Status       | Priority | Category   | Version     | Component | Replies | Last updated   | Assigned to | Created       |
|--|--------------|----------|------------|-------------|-----------|---------|----------------|-------------|---------------|
| Caption is not working with 'Title text' and 'Alt text' properties. <a href="#">new</a>                            | Active       | Normal   | Bug report | 8.x-1.x-dev | Code      | 1 1 new | 13 min 27 sec  |             | 13 min 27 sec |
| Replace deprecated usage of entity_load('image_style') with a direct call to ImageStyle::load. <a href="#">new</a> | Needs review | Normal   | Task       | 8.x-1.x-dev | Code      | 5 5 new | 4 hours 54 min |             | 5 hours 7 min |

- 2.** On the next screen we will see a form. We will fill in all the required fields: **Title, Category** as Bug report, **Version** as 8.x-1.x-dev, **Component** as Code, and the **Issue summary** field. Once I submitted my form, an issue was created at <https://www.drupal.org/node/2645160>. You should see an issue on Drupal (<https://www.drupal.org/>) like this:

The screenshot shows a Drupal module page for 'Colorbox'. The main title is 'Caption is not working with 'Title text' and 'Alt text' properties.' Below it, there are 'View' and 'Edit' buttons. A note states: 'Caption is not working with 'Title text' and 'Alt text' properties, when used with the File Entity (file\_entity) module.' A section titled 'How to reproduce:' lists steps to fix the issue. Below that is an 'Upload new files' button. A 'Comments' section shows a single comment from 'neerajskydive' with a link '#1' and a 'report as spam' button. To the right, a sidebar titled 'Active' displays issue details: Project: Colorbox, Version: 8.x-1.x-dev, Component: Code, Priority: Normal, Category: Bug report, Assigned: Unassigned, Reporter: neerajskydive, Created: January 6, 2016 - 18:06, Updated: January 6, 2016 - 18:06. A green 'Update this issue' button is at the bottom.

- 3.** Next, the Maintainers of the colorbox module will look into this issue and reply accordingly. Actually, @Frjо replied saying "I have never used that module but if someone who does sends in a patch I will take a look at it." He is a contributor to this module, so we will wait for some time and will see if someone can fix this issue by giving a patch or replying with useful comments. In case someone gives the patch, then we have to apply that to the colorbox module. This information is available on Drupal at <https://www.drupal.org/patch/apply>.

## What just happened?

We understood and created an issue in the Colorbox module's issue queue list. We also looked into what the required fields are and how to fill them to create an issue in the Drupal module queue list form.

## **Summary**

In this chapter, we looked at a way to use our d8dev site with multimedia, creating image styles using some custom code, and learned some new ways of interacting with the Drupal developer community. We also worked with the Colorbox module to add images to our d8dev content with the Colorbox pop-up feature. Lastly, we looked into the custom module to work with custom CSS files.

In the next chapter, we will revisit the Colorbox module with some enhancements, and add some features to our d8dev site that will enable visitors to our site to provide feedback and interact with our site's content.

# 8

## How Does It Taste? – Getting Feedback

*This chapter will walk through the code that adds a placeholder for HTML form elements in the contact module. We will also revisit the Colorbox module that we installed in the previous chapter. We will make some enhancements to the code (an advanced real-world example), and walk through the process of working with patches. We will also implement the Review and Like system with recipe content type using comment types and views configurations.*

### Introduction to the Drupal contact form

A very simple contact form is included with Drupal core and is enabled by default. The core contact form provides a good starting point to introduce some interactive features to our d8dev site.

#### Time for action – enabling and configuring the core contact form

We will configure the core contact form so that anonymous visitors to our site will be able to provide feedback about the site:

1. In Drupal 8, the contact module is enabled by default. We just need to set permissions to the **ANONYMOUS** and **AUTHENTICATED** roles to use the feedback form.

*How Does It Taste? – Getting Feedback*

---

2. Now, open the d8dev site in your favorite browser, and click on the **Extend** link in the **Admin** toolbar. Scroll down to the module named **Contact** under the **Core** group; you will notice that the module is enabled. Then, click on the **Permissions** link:

A screenshot of a web page showing the 'Contact' module configuration. At the top left is a checked checkbox next to 'Contact'. To its right is a brief description: 'Enables the use of both personal and site-wide contact forms.' Below this are two lines of text: 'Machine name: contact' and 'Version: 8.0.1'. At the bottom of the page are three links: 'Help', 'Permissions', and 'Configure'.

3. On the **Permissions** page, you will notice that the **Use the side-wide contact form** permission is enabled for the **ADMINISTRATOR**, **ANONYMOUS**, and **AUTHENTICATED** roles by default. So, anonymous users can use the contact form.
4. Next, log out and navigate to `http://localhost/contact/feedback`. You will see a simple contact form, as shown in the following screenshot:

A screenshot of a contact form titled 'Website feedback'. The form has four fields: 'Your name \*' (text input), 'Your email address \*' (text input), 'Subject \*' (text input), and 'Message \*' (large text area). At the bottom are two buttons: 'Send message' and 'Preview'.

## What just happened?

We enabled a simple contact form to get feedback from the visitors to our d8dev site.

## Adding placeholder text to our contact form

The core Contact module provides a decent out-of-the-box contact form for our site. But what if we want to customize it a bit? Say, for example, we want to add some placeholder text to explain to the users why they should fill out the form. As we have seen in previous chapters, there are usually multiple ways to accomplish customizations like this with Drupal.

## Using configurations to add placeholder text to the contact form

We will configure the core contact form to enable placeholders to explain to the users why they should fill out the form.

### Time for action – adding placeholder text to our site contact form

The core Contact module provides a decent out-of-the-box contact form for our site. The core also provides configurable placeholders for fields. So, we add the placeholders for form fields to display placeholders to explain to the users why they should fill out the form:

1. Open up the d8dev site in your favorite browser, and click on the **Structure** link in the **Admin** toolbar. Click on **Contact forms** on the next page you can see **Website feedback** listed.
2. Now, click on **Manage form display** in the **OPERATIONS** column for the **Website feedback** form:

| + Add contact form    |               |          |   |
|-----------------------|---------------|----------|---|
| FORM                  | RECIPIENTS    | SELECTED | OPERATIONS  |
| Personal contact form | Selected user | No       | <a href="#">Manage fields</a>   |
| Website feedback      | info@d8.com   | Yes      | <a href="#">Edit</a><br><a href="#">Manage fields</a><br><b>Manage form display</b> (highlighted)<br><a href="#">Manage display</a><br><a href="#">Delete</a> |

*How Does It Taste? – Getting Feedback*

- 3.** On the next screen, you can see that the **Subject** and **Message** fields are configurable:

The screenshot shows the 'Manage form display' page for a contact form. It lists several fields with their current widget settings:

| FIELD               | WIDGET                    |  |
|---------------------|---------------------------|--|
| Sender name         | Visible                   |  |
| Sender email        | Visible                   |  |
| Subject             | Textfield                 | textfield size: 60  |
| Message             | Text area (multiple rows) | Number of rows: 12  |
| Send copy to sender | Visible                   |  |

Below the table, there's a section for 'Disabled' fields and a note that 'No field is hidden.' At the bottom is a blue 'Save' button.

- 4.** Now, click on the settings icon for the **Subject** field, fill in **Placeholder** as **Please enter your subject here!**, and click on the **Update** button:

The screenshot shows the 'Widget settings' dialog for the 'Subject' field. It includes the following fields:

- Widget settings: **Textfield**
- Size of textfield \***: 60
- Placeholder**: Please enter your subject here!
- A descriptive note: Text that will be shown inside the field until a value is entered. This hint is usually a sample value or a brief description of the expected format.
- Buttons: **Update** (highlighted in blue) and **Cancel**.

- 5.** Next, click on the settings icon for the **Message** field, fill in **Placeholder** as **Please enter your Message here!**, change the **Rows** value from 12 to 8, and click on the **Update** button:

The screenshot shows the 'Widget settings' dialog for the 'Message' field. It includes the following fields:

- Widget settings: **Text area (multiple rows)**
- Rows \***: 8
- Placeholder**: Please enter your Message here!
- A descriptive note: Text that will be shown inside the field until a value is entered. This hint is usually a sample value or a brief description of the expected format.
- Buttons: **Update** (highlighted in blue) and **Cancel**.

6. Next, log out and navigate to `http://localhost/contact/feedback` as an **ANONYMOUS** user. You will see a simple contact form, as shown in the following screenshot:

The screenshot shows a web-based contact form titled "Website feedback". It contains four input fields: "Your name \*", "Your email address \*", "Subject \*", and "Message \*". Each field has a placeholder text: "Please enter your subject here!" for the Subject field and "Please enter your Message here!" for the Message field. Below the form are two buttons: "Send message" and "Preview".

[  You can also try the Contact Storage module, which provides storage for Contact messages; these are full-fledged entities in Drupal 8. This module provides features such as Message storage, Edit messages, and Delete messages. For more details, check out [https://www.drupal.org/project/contact\\_storage](https://www.drupal.org/project/contact_storage). ]

### **What just happened?**

We configured the **Placeholder** text for the **Subject** and **Message** fields in the **Manage form display** page for the contact form.

## Using custom code to add placeholder text to the Name and Email fields

Drupal provides configurable placeholders for the **Subject** and **Message** fields but not for the **Name** and **Email** fields, so we add placeholders for them using custom code in our d8dev module.

### Time for action – adding placeholder text to Name and Email fields

We will utilize the core `hook_form_alter` hook to alter the core contact form to display placeholders for name and e-mail fields. We will be adding this code to the `d8dev.module` file:

1. In PhpStorm, open the `d8dev.module` file in your custom module at `/modules`.
2. Next, switch over to the browser with the core contact form loaded, and find the `form_id` of the form element. Right-click on the page and click on **View source code**. On the next page, you can see HTML code. Now search for `form_id` and you will find that `contact_message_feedback_form` is the required form ID.
3. In another way, we can use the `drupal_set_message()` core function to find the `form_id`. To do this, we need to add the following code on top of the `d8dev.module` file:

```
use Drupal\Core\Form\FormStateInterface;

/**
 * Implements hook_form_alter().
 */
function d8dev_form_alter(&$form, FormStateInterface $form_state,
$form_id) {
  drupal_set_message($form_id);
}
```

4. Now, reload the `http://localhost/contact/feedback` page as a logged-in user, and you will see a message displaying `form_id`, as follows:

The screenshot shows a web-based contact form titled "Website feedback". The form has a light green header bar with the title. Below it, there's a breadcrumb navigation "Home > Contact". On the left, there's a sidebar with "Search" and "Tools" sections, and a "Add content" button. The main form area has fields for "Your name" (filled with "krishna"), "Your email address" (filled with "info@d8.com"), "Subject" (placeholder: Please enter your subject here!), and "Message" (placeholder: Please enter your Message here!). At the bottom, there are buttons for "Send message" and "Preview".

5. Next, we need to add a placeholder HTML attribute to the `name` and `mail` form elements, as follows:

```
$form['name']['#attributes'] = array('placeholder' =>
array('Please enter your first and last name.'));
$form['mail']['#attributes'] = array('placeholder' =>
array('Please enter your e-mail address.'));
```

6. Add the preceding placeholder code only for the contact form. Now our `d8dev_form_alter()` function looks like this:

```
/**
 * Implements hook_form_alter().
 */
function d8dev_form_alter(&$form, FormStateInterface
$form_state, $form_id) {
  if ($form_id == 'contact_message_feedback_form') {
    $form['name']['#attributes'] = array('placeholder' =>
array('Please enter your first and last name.'));
    $form['mail']['#attributes'] = array('placeholder' =>
array('Please enter your e-mail address.'));
  }
}
```

- 7.** Next, clear the cache by going to **Administration | Configuration | Development | Performance**. Now reload the contact page as an **ANONYMOUS** user. Then you will see the following form:

The screenshot shows a web form titled "Website feedback". It contains four fields: "Your name \*", "Your email address \*", "Subject \*", and "Message \*". Each field has a placeholder text: "Please enter your first and last name.", "Please enter your e-mail address.", "Please enter your subject here!", and "Please enter your Message here!". Below the form are two buttons: "Send message" and "Preview".

### **What just happened?**

We used the `hook_form_alter` hook to add placeholder text to the core contact form for the **Name** and **Email** fields.

### **Time for another recipe!**

Just in case you were getting hungry, we are going to add a new recipe. Let's add it now so that we have something delicious to eat as we work our way through the rest of this challenging chapter! The recipe for this chapter is garlic chicken Indian style. Enjoy!



- ◆ **Name:** Garlic chicken Indian style
- ◆ **Description:** Garlic chicken recipe Indian style, Learn how to make garlic chicken in 20 minutes, Includes restaurant style and home style starters, curries, biryanis and more. This garlic chicken can be prepared in just 20 mins excluding the marinating time. It makes a perfect starter or a side to make a meal exotic.
- ◆ **recipeYield:** Four servings
- ◆ **prepTime:** 20 minutes
- ◆ **cookTime:** 20 minutes
- ◆ **Ingredients:**
  - 300 grams of chicken strips or boneless cubes
  - 4 tbsp. yogurt/dahi/curd (do not use sour yogurt)
  - Salt—very little
  - A generous pinch of pepper
  - Two tbsp. oil
  - One sprig curry leaf
  - $\frac{1}{4}$  tsp cumin
  - One onion chopped finely
  - Five red chilies or  $\frac{3}{4}$  tsp red chili powder
  - Five garlic cloves crushed
  - $\frac{3}{4}$  tbsp. vinegar
  - $\frac{1}{4}$  tsp sugar (optional)

- ❑ Salt to taste
- ❑ Water—15 to 30 ml (the lesser the better) to make a paste
- ❑ A few coriander leaves
- ❑ Onion wedges or rings
- ❑ Lemon

◆ **Instructions:**

1. Mix yogurt, salt, and pepper with  $\frac{1}{2}$  to  $\frac{3}{4}$  cup of water. Beat well and set aside.
2. Wash chicken and soak it in this prepared butter milk for 2 hours to overnight in the refrigerator.
3. Soaking for at least 6 hours is recommended to get soft chicken.
4. Blend and make a paste of the ingredients mentioned for chili garlic sauce.
5. Drain the buttermilk and add the prepared sauce.
6. Mix well and leave it for 5 minutes. If you want to prepare it at a later time, you can put it back in the refrigerator.
7. Heat a pan with oil. Add cumin and curry leaves. Sauté until the cumin begins to splutter.
8. Add onions and fry until golden.
9. Add the chicken and fry on high heat for 2 to 3 minutes. When you see the chicken turning white, cover it with a lid, lower the heat, and cook until the chicken turns tender. This barely takes 5 minutes if you are using strips.
10. Remove the lid and fry until the moisture evaporates and the sauce begins to cling to the chicken. The longer it is roasted, the more aromatic and tastier the sauce becomes, but the chicken tends to lose its tenderness.
11. So take off from the heat when it is of your liking. Take care not to burn; garlic burns faster.

## Colorbox file enhancements

We added the Colorbox module to our d8dev site in the previous chapter to display the images in the overlay. Currently, the Colorbox module displays the title of the recipe as the caption for all recipe images displayed in the Colorbox overlay for our Recipe content type, as shown in the following screenshot. But it's not working with image properties such as title and alt values.



So for that, we raised an issue in the Colorbox issues queue (<https://www.drupal.org/node/2645160>) in *Chapter 7, Adding Media to Our Site*. Colorbox maintainer @frj0 replied as follows:

*"The file entity module is a contrib module so what you really is asking for is that Colorbox support it. Change the title so this is clear. I have never used that module but if someone who does sends in a patch I will take a look at it."*

Before we begin creating a patch, we need to have a clear understanding of why it isn't showing up in this case of the `title` or `alt` property. I went through the Colorbox module and did debug on all preprocess functions where it was rendering the images field properties. I found that the `template_preprocess_colorbox_formatter()` function is the right function that loads all attributes of the image field. But when the file entity module is enabled, it does not load because the files have become entities and Colorbox doesn't know how to render from image files. So, we are going to create a patch for this issue and submit it in the Colorbox issue that we created in the last chapter.

## Time for action – enhancing the Colorbox module with image title and alt captions

So, where do we begin implementing this proposed approach and complete the associated feature request? We will start by debugging the available processed \$variable in the `template_preprocess_colorbox_formatter()` function so that we can modify or add required properties to the image field:

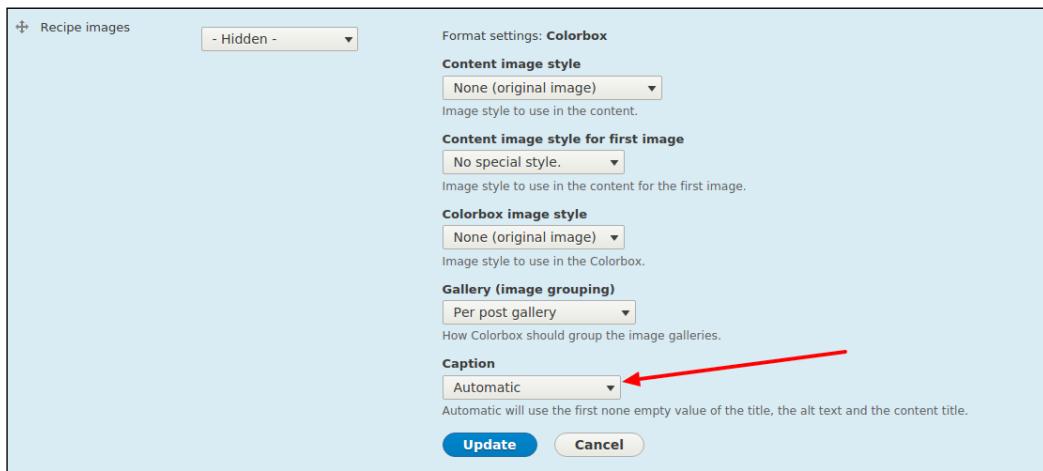
1. In PhpStorm, open the `colorbox.inc` file in your contribute modules at the `/modules` folder.
2. Before we start modifying or adding code, let's understand the code in the `template_preprocess_colorbox_formatter()` function in the `colorbox.inc` file. First, we need to know where caption settings are saved. There is the `$settings` variable that has been declared; add the following code next to the `$settings` variable (line 35):

```
var_dump($settings); die;
```

3. Clear the cache from the **Performance** page and reload the recipe content; you will see the following:

```
array (size=8)
  'colorbox_node_style' => string '' (length=0)
  'colorbox_node_style_first' => string '' (length=0)
  'colorbox_image_style' => string '' (length=0)
  'colorbox_gallery' => string 'post' (length=4)
  'colorbox_gallery_custom' => string '' (length=0)
  'colorbox_caption' => string 'auto' (length=4) ←
  'colorbox_caption_custom' => string '' (length=0)
  'style_first' => boolean false
```

4. We noticed that auto is the colorbox caption for this content type. To confirm this, go to **Structure | Content types** and click on **Manage display** for the recipe content type. Next, click on the settings icon for the **Recipe images** field and you will notice the **Caption as Automatic**:



5. Your settings might be different, so in the last step, you can see relatively.  
6. In the next lines, we can see a switch statement declared as follows to set the \$caption variable, which is going to be displayed as caption in the Colorbox overlay:

```
switch ($settings['colorbox_caption']) {
  case 'auto':
    // If the title is empty use alt or the entity title in
    // that order.
    if (!empty($item->title)) {
      $caption = $item->title;
    }
}
```

```

elseif (!empty($item->alt)) {
    $caption = $item->alt;
}
elseif (!empty($entity_title)) {
    $caption = $entity_title;
}
else {
    $caption = '';
}
break;
case 'title':
$caption = $item->title;
break;
case 'alt':
$caption = $item->alt;
break;
case 'entity_title':
$caption = $entity_title;
break;
case 'custom':
$token_service = \Drupal::token();
$caption = $token_service-
>replace($settings['colorbox_caption_custom'],
array($entity_type => $entity, 'file' => $item),
array('clear' => TRUE));
break;
default:
$caption = '';
}

```

7. So, we can understand that the `$caption` variable needs to be modified to set alt or title text as the caption. To confirm this, when `$caption` is getting loaded in the colorbox overlay; we just have to set some dummy text to it and we will see the effect on the recipe content. Before adding the dummy text, remove the `var_dump($settings); die;` line you added in the last step.
8. Next, add the following line next to the switch statement:

```
$caption = 'This is test caption!';
```

- 9.** Clear the cache from the **Performance** page. Reload the recipe content and click on any image to check the caption; you will see the following screenshot:



**10.** Before moving to the next step, remove the dummy caption code you added in the last step: `$caption = 'This is test caption!';`.

**11.** Next, preprocess the title of the `attributes` variable, which needs to be loaded from the image file entity. Add the following code before the `$variables['attributes']['title'] = $caption;` line, which is at the end of the `template_preprocess_colorbox_formatter()` function:

```
// if File Entity module is enabled, load attribute values
// from file entity.
if (\Drupal::moduleHandler()->moduleExists('file_entity')) {
    // file id of the save file.
    $fid = $item->target_id;
    // load file object
    $file_obj = file_load($fid);
    $file_array = $file_obj->toArray();
    // populate the image title
    if (Unicode::strlen($file_array['field_image_title_text'])
        [0]['value']) != 0 && empty($item->title) &&
    $settings['colorbox_caption'] == 'title') {
        $caption =
            $file_array['field_image_title_text'][0]['value'];
    }
}
```

```

    }
    // populate the image alt text.
    if (!empty($file_array['field_image_alt_text'][0]['value'])
&& empty($item->alt) && $settings['colorbox_caption'] ==
'alt') {
    $caption =
    $file_array['field_image_alt_text'][0]['value'];
}
}

```

- 12.** First, we checked whether the `file_entity` module is enabled because this module is not dependent on the `colorbox` module. The purpose of the `file_entity` module is to provide interfaces for managing files and files to be fieldable as entities, so the properties of the image file can be rendered as objects. Next, we added code that loads the file object and we set the `$caption` variable by validating the title and alt text properties from the `$file` object.
- 13.** To see this effect on the recipe content, clear the cache from the Performance page and reload the recipe content before making sure that your image has updated with alt and title texts.
- 14.** Next, go to the Recipe content type manage display page, click on the settings icon for the **Recipe images** field and update the **Caption** as **Title text**. Click on the **Save** button at the bottom of the **Manage display** page and reload the recipe content page to see the changes. Click on any image. WOW! Now it's showing title text as captions (see the next screenshot). Alternatively, update **Caption** as **Alt text** in the settings and check the recipe content; it will work:



## What just happened?

We updated the colorbox module with custom code to work with the title and alt text properties of images.

## Contributing our code to Drupal

Now we have code that will be usable across the colorbox module. But, how do we get this code into the hands of other Drupal users, and better yet, have it maintained as part of the colorbox module? To share this code, we are going to create a patch.

### Time for action – creating a patch and uploading it on the Drupal issues queue

Before we can create a patch the Drupal 8/Git way, we need to check out the colorbox module project with Git to a new directory:

1. Go to <https://www.drupal.org/project/colorbox/git-instructions>. At the top, you can see the **Version to work from** select box. Select **8.x-1.x** and click on the **Show** button:

The screenshot shows the 'Colorbox' project page under the 'Download & Extend' tab. The 'Version control' section is highlighted with a red arrow pointing to the 'Version to work from' dropdown menu. The dropdown is set to '8.x-1.x'. A second red arrow points to the 'Show' button next to the dropdown. The page also features a 'Maintainers for Colorbox' sidebar listing contributors like fjo, Sam152, podarok, grisendo, jdwfly, and others, along with their commit counts and last activity times.

- 2.** Now open the Terminal and go to any location (the `cd <path/to/go>` command) where you wish to clone the colorbox module. Run the following command to clone the colorbox Git repository:

```
$ git clone --branch 8.x-1.x https://git.drupal.org/project/
colorbox.git

Cloning into 'colorbox'...
remote: Counting objects: 2082, done.
remote: Compressing objects: 100% (1545/1545), done.
remote: Total 2082 (delta 1253), reused 661 (delta 410)
Receiving objects: 100% (2082/2082), 1.84 MiB | 27.00 KiB/s, done.
Resolving deltas: 100% (1253/1253), done.

Checking connectivity... done.
```

- 3.** Next, we need to make the same changes in the `colorbox.inc` file that we just made for our d8dev site to the code that we just cloned with Git.
- 4.** Now that we have updated the code, we are ready to create our patch. Following the Drupal guidelines at <http://drupal.org/node/707484>, run this command from the colorbox root folder:

```
$ git diff > 2645160-6.patch
```

- 5.** Next, we will comment on the issue at <http://drupal.org/node/2645160> and upload our patch with our comments, making sure that the status of the issue is set to needs review:

The screenshot shows a Drupal issue page for issue #6. The user neerajskydiver has commented a day ago and uploaded a patch named "2645160-6.patch". The patch is 1.19 KB in size and has been tested on PHP 5.5 & MySQL 5.5, with D8.1 2 pass, 1 fail results. A link to "Add test" is also present. Below the patch, there is a note: "Optimized and updated patch as per latest branch commit." A "report as spam" link is at the bottom right.

| Status | File                            | Size    |
|--------|---------------------------------|---------|
| new    | <a href="#">2645160-6.patch</a> | 1.19 KB |

If we want to apply the patch just shown, we use the following `git` command:

```
git apply -v [patchname.patch]
```

If we want to revert the last applied patch, then we use these `git` commands:

```
git checkout [filename]
git reset --hard
```

## **What just happened?**

Not only did we update our code so that it will work across the Colorbox module and any number of Drupal sites, but we also helped the Drupal community. We learned how to create a patch. We will get back to the Colorbox issue at <http://drupal.org/node/2645160> to see how the Drupal community responds to our patch.

## **Recipe reviews with comments**

Now that we have enhanced the Colorbox module with configurable captions, we will turn our attention back to enhancing user interaction for the site. One great way to get visitors to interact with a website is to allow them to review and like/dislike the content, in this case, recipes.

### **Time for action – configuring comments as recipe reviews**

We will now set up the comments so that visitors to our d8dev site will be able to review the recipes:

- 1.** We will configure our Recipe content type to use the comments. Open up the d8dev site in your favorite browser, click on the **Structure** link in the **Admin** toolbar, then click on the **Comment types** link, and finally click on the **Add content type** link for your Recipe content type.
- 2.** On the next screen, enter **Recipe Review** as the value for the **Label** field and **Content** as the **Target entity type** field. Click on the **Save** button:

**Add comment type** 

Home » Administration » Structure » Comment types

**Label \***

 Machine name: recipe\_review [Edit]

**Description**

Describe this comment type. The text will be displayed on the *Comment types* administration overview page

**Target entity type**

Content ▾  
The target entity type can not be changed after the comment type has been created.

**Save**

3. We added a new comment type and it can be used in any content types. Click on the **Structure** link in the **Admin** toolbar, then click on the **Content types** link, and finally click on the **Manage fields** link for your Recipe content type.
4. Next, click on the **Add field** link. On the next screen, select **Comments** as **Add a new field**, enter Recipe Review as **Label**, and click on the **Save and continue** button:

**Add field** 

Home » Administration » Structure » Content types » Recipe » Manage fields

**Add a new field**      **Re-use an existing field**

Comments ▾ or - Select an existing field - ▾

**Label \***

 Machine name: field\_recipe\_review [Edit]

**Save and continue**

5. On the next screen, select **Recipe Review** as **Comment type** and click on the **Save field settings** button. Next, leave all fields as default and click on the **Save settings** button:

The screenshot shows the 'Field settings' page for the 'Recipe Review' content type. At the top, there are 'Edit' and 'Field settings' tabs. Below them, a breadcrumb navigation shows: Home » Administration » Structure » Content types » Recipe » Manage fields » Recipe Review. A note states: 'These settings apply to the *Recipe Review* field everywhere it is used. These settings impact the way that data is stored.' Under 'Comment type \*', a dropdown menu is open, showing 'Recipe Review' with a red arrow pointing to it. A note below says: 'Select the Comment type to use for this comment field. Manage the comment types from the [administration overview page](#).' At the bottom is a blue 'Save field settings' button.

6. We have enabled comments for recipe content types. Now we reload the recipe content as an **ANONYMOUS** user to see the default comments form. But we cannot see the form because of permission issues. To set the permission, go to **people | permissions**, check the **Post comments** permission for **ANONYMOUS** users, scroll down to the bottom of the page, and click on the **Save permissions** button. Again, reload the recipe content as an **ANONYMOUS** user and you will see something like the following screenshot:

Add new comment

Your name

Subject

**Comment \***

• Allowed HTML tags: <a href hreflang> <em> <strong> <cite> <blockquote cite> <code> <ul type> <ol start type> <li> <dl> <dt> <dd> <h2 id> <h3 id> <h4 id> <h5 id> <h6 id>

• Lines and paragraphs break automatically.

• Web page addresses and email addresses turn into links automatically.

7. We don't need the name, subject, and comment fields provided by default. We are going to delete them and add the **e-mail** and **How does it taste ?** fields. Now click on **Structure** in the **Admin** bar, and then click on the link **Comment types**.
8. Now, click on **Manage fields** from the Operations for Recipe review comment type. You can see that only the **Comment** field is there. Click on the **Delete** link from **OPERATIONS** for the **Comment** field. On the next screen, click on the **Delete** button to delete this field:

**Manage fields** ☆

[Edit](#) [Manage fields](#) [Manage form display](#) [Manage display](#) [Devel](#)

Home » Administration » Structure » Comment types » Edit

[+ Add field](#)

| LABEL   | MACHINE NAME | FIELD TYPE             | OPERATIONS  |
|---------|--------------|------------------------|---|
| Comment | comment_body | Text (formatted, long) | <a href="#">Edit</a><br><a href="#">Storage settings</a><br><b><a href="#">Delete</a></b> |

- 9.** Next, click on **Add field**, select **Add a new field** as **Email**, enter **Label** as **E-mail**, and click on the **Save and continue** button:

The screenshot shows the 'Add field' configuration page. At the top, there's a breadcrumb navigation: Home > Administration > Structure > Comment types > Edit > Manage fields. Below this, there are two main options: 'Add a new field' (selected) and 'Re-use an existing field' (disabled). Under 'Add a new field', a dropdown menu shows 'Email' is selected. There's also a 'Label \*' field containing 'E-mail' and a note below it stating 'Machine name: field\_e\_mail [Edit]'. A blue 'Save and continue' button is at the bottom of the form.

- 10.** On the next screen, leave the default values **Limited**, **1**, and click on the **Save field settings** button. Next, check **Required field** and click on the **Save settings** button.
- 11.** Now, click on the **Add field** link. Select **Text (plain, long)** as **Add a new field** and **How does it taste ?** as the **Label**. Click on the **Save and continue** button.
- 12.** On the next screen, leave the default settings as **1** and click on the **Save field settings** button. Again on the next screen, leave all fields as default and click on the **Save settings** button.
- 13.** Next, reload the recipe content as an **ANONYMOUS** user. We can see the form as shown in the following screenshot:

Add new comment

Your name

Subject

E-mail \*

How does it taste ?

14. But we can still see the **Your name** and **Subject** fields, which are not required. So we will disable them from the **Manage form display** page. Now, click on **Manage form display** for the Recipe Review comment type. On the next screen, select the **Hidden** value for the **Author** and **Subject** fields and click on the **Save** button. See the following screenshot:

Manage form display ☆

Edit Manage fields Manage form display Manage display Devel

Home » Administration » Structure » Comment types » Edit Show row weights

| FIELD               | WIDGET                    |                   |
|---------------------|---------------------------|-------------------|
| E-mail              | Email                     | No placeholder    |
| How does it taste ? | Text area (multiple rows) | Number of rows: 5 |
| <b>Disabled</b>     |                           |                   |
| Author              | - Hidden -                | ←                 |
| Subject             | - Hidden -                | ←                 |

- 15.** Now we reload the recipe content, and we can see the comment form as follows:

The screenshot shows a 'Comment' form titled 'Add new comment'. It contains an 'E-mail \*' field (empty), a large text area for 'How does it taste ?' (empty), and two buttons at the bottom: 'Save' and 'Preview'.

- 16.** We need to restrict viewing comments for **ANONYMOUS** and **AUTHENTICATED** user roles since we need to display only the number of likes for the recipe content. To set the permission, go to **people | permissions**. Uncheck the **View comments** permission for **ANONYMOUS** and **AUTHENTICATED** users, scroll down to the bottom of the page, and click on the **Save permissions** button.

### **What just happened?**

We created a new comment type, Recipe review, and enabled it in the Recipe content type. Also, we configured new fields in the Recipe review comment type.

## **Time for action – enhancing the liking system using comments and views**

We will now set up a like/dislike button so that visitors to our d8dev site will be able to like/dislike the recipes. Also, we will show how many people have liked each recipe content:

- 1.** Firstly, we will add the **Do you like it ?** button in the Recipe Review comment type so that visitors to our d8dev site will be able to like/dislike the recipes. Open up the d8dev site in your favorite browser, and click on the **Structure** link in the **Admin** toolbar. Then click on the **Comment types** link, and finally click on the **Manage fields** link for your Recipe review in the **Operations** dropdown.
- 2.** Click on the **Add field** link to add a new field. On the next screen, select **List (integer)** as **Add a new field** and **Do you like it ?** as **Label**. Click on the **Save and continue** button:

**Add field** ☆

Home » Administration » Structure » Comment types » Edit » Manage fields

**Add a new field**      **Re-use an existing field**

List (integer)      or      - Select an existing field -

**Label \***  
Do you like it ?      Machine name: field\_do\_you\_like\_it\_ [Edit]

**Save and continue**

3. Now, enter the following text in the **Allowed values list** field, leave other settings as default, and click on the **Save field settings** button:

1 | Yes  
0 | No

4. For this step, check the **Required field** button, select **Default value** as **Yes**, and click on the **Save settings** button.
5. Now click on the **Manage form display** link. Change the **Do you like it ?** field widget to the **Check boxes/radio buttons** widget and click on the **Save** button:

**Manage form display** ☆

Edit      Manage fields      **Manage form display**      Manage display      Devel

Home » Administration » Structure » Comment types » Edit

| FIELD               | WIDGET                    | Settings                       |
|---------------------|---------------------------|--------------------------------|
| E-mail              | Email                     | No placeholder                 |
| How does it taste ? | Text area (multiple rows) | Number of rows: 5              |
| Do you like it ?    | Check boxes/radio buttons | (highlighted with a red arrow) |
| <b>Disabled</b>     |                           |                                |
| Author              | - Hidden -                |                                |
| Subject             | - Hidden -                |                                |

**Save**

6. Next, reload the recipe content and you can see the comment form as follows:

The screenshot shows a web-based comment form titled "Add new comment". The form includes fields for "E-mail \*", "How does it taste?", "Do you like it? \*", and two radio button options ("Yes" or "No"). At the bottom are "Save" and "Preview" buttons.

Add new comment

E-mail \*

How does it taste ?

Do you like it? \*

Yes

No

Save Preview

7. We have enabled the **Like** button. Next, we need to show how many people have liked the recipe content. We are going to create one view block to display the count of likes for each recipe content. Open up the d8dev site in your favorite browser. Click on the **Structure** link in the **Admin** toolbar, then click on the **Views** link, and finally click on the **Add new view** link.
8. On the **Add new view** page, enter the view name as **Recipe likes count**. In the **View settings** section, select the **Show** field as **Comments** and the **of type** field as **Recipe review**. In the **Block settings** section, check **Create block** and click on the **Save and edit** button.
9. Next, in the **Format** section, click on the **Show: Comment** link. Then, a popup will appear. Change the **Row** field value comment to fields and click on the **Apply (all displays)** button:

**Displays**

Display name: Block

**TITLE**  
Title: Recipe likes count

**FORMAT**  
Format: Unformatted list | Settings  
Show: **Comment** | Default

**FIELDS**  
The selected style or row format does not use fields.

**FILTER CRITERIA**  
Comment: Approved status (= Yes)  
(Content) Content: Publishing status (= Yes)  
Comment: Comment Type (= Recipe Review)

**SORT CRITERIA**

**BLOCK SETTINGS**  
Block name: None  
Block category: Lists (Views)  
Allow settings: Items per page  
Access: Permission | View comments

**HEADER**

**FOOTER**

**NO RESULTS BEHAVIOR**

**PAGER**  
Use pager: Display a specified number  
More link: No  
Link display: None

- 10.** Next, in the **Advanced** section and under the **Other** group, click on the **Use aggregation: No** link. Then, a popup will appear. Check **Aggregate** and click on the **Apply (all displays)** button:

Duplicate Block ▾

**▼ ADVANCED**

CONTEXTUAL FILTERS

RELATIONSHIPS

Content

EXPOSED FORM

Exposed form style: Basic | Settings

**OTHER**

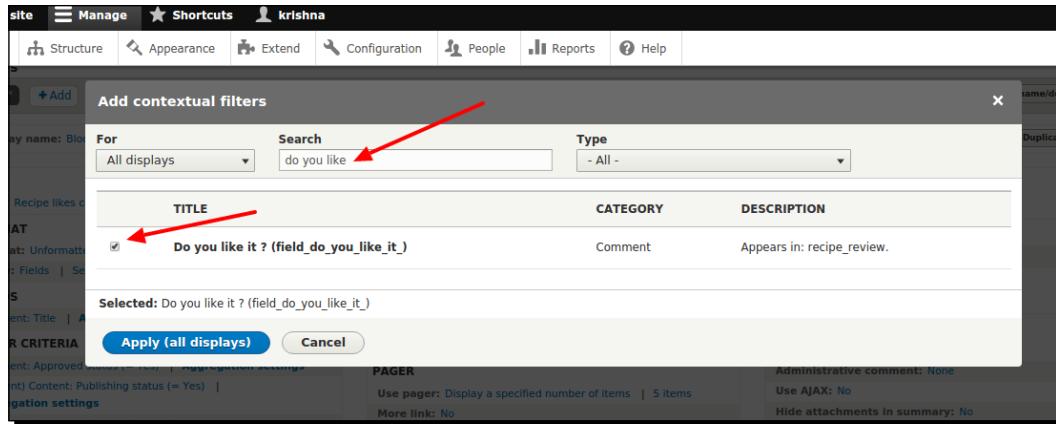
Machine Name: block\_1  
Administrative comment: None  
Use AJAX: No  
Hide attachments in summary: No  
Contextual links: Shown  
**Use aggregation: No**

Query settings: Settings  
Caching: Tag based  
CSS class: None  
Hide block if the view output is empty: No

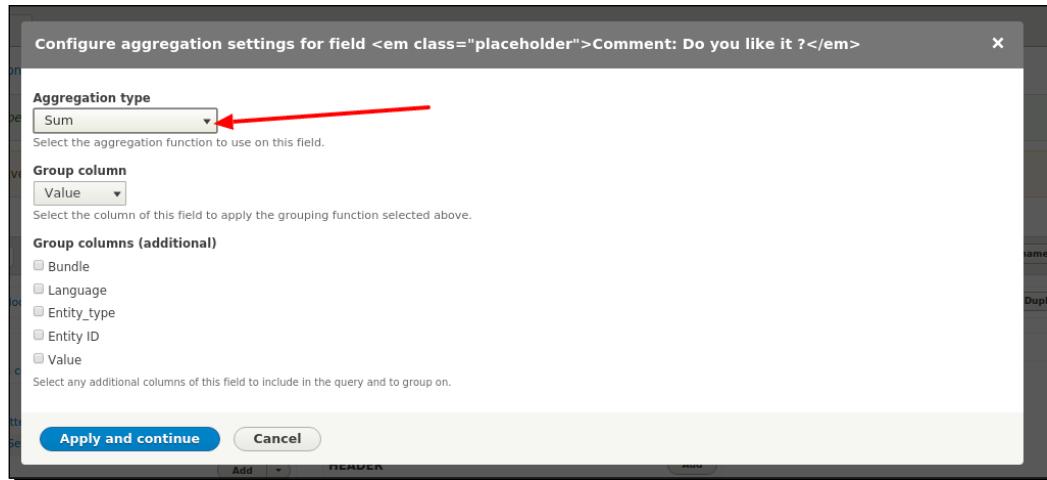
*How Does It Taste? – Getting Feedback*

---

- 11.** Now, in the **Fields** section, click on the **Add** link. In the popup, search for the keywords `do you like`. Then check the **Do you like it ? (field\_do\_you\_like\_it\_)** checkbox and click on the **Apply(all displays)** button:



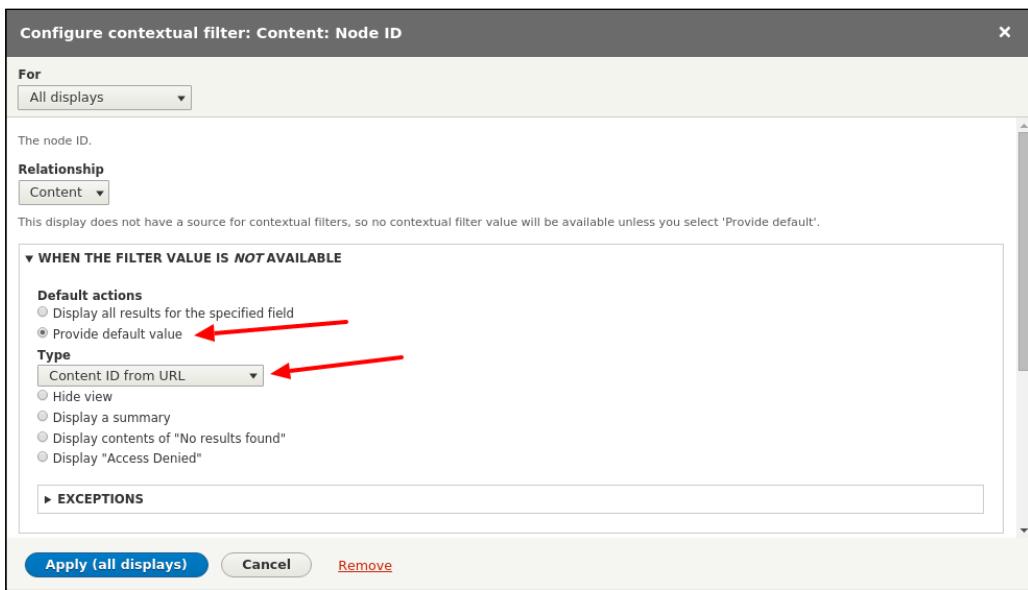
- 12.** On the next screen, select **Sum** as the **Aggregation type** field and click on the **Apply and continue** button:



- 13.** On the next screen, scroll down to the **REWRITE RESULTS** section and click on it. Check the **Override the output of this field with custom text** checkbox and enter the following text in the **Text** field:

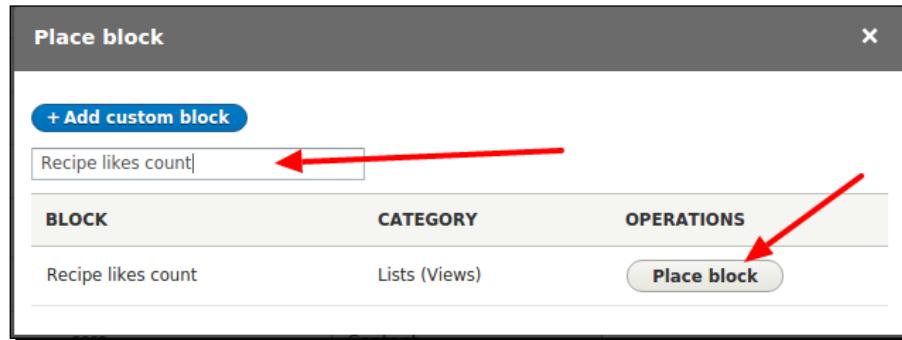
```
{{ field_do_you_like_it_ }} likes
```

- 14.** Next, in the **STYLE SETTINGS** section, check the **Customize field HTML** checkbox. Select the **HTML element** as **STRONG** and click on the **Apply(all displays)** button.
- 15.** Now, in the **ADVANCED** section, click on the **Add** link for **CONTEXTUAL FILTERS**. In the popup, search for the keywords **node id**. Then check the **Node ID** checkbox and click on the **Apply(all displays)** button. On the next screen, leave the settings as default and click on the **Apply and continue** button.
- 16.** Under the **WHEN THE FILTER VALUE IS NOT AVAILABLE** section on the next screen, select the **Provide default value** radio button. Select **Type** as **Content ID from URL** and click on the **Apply(all displays)** button at the bottom:

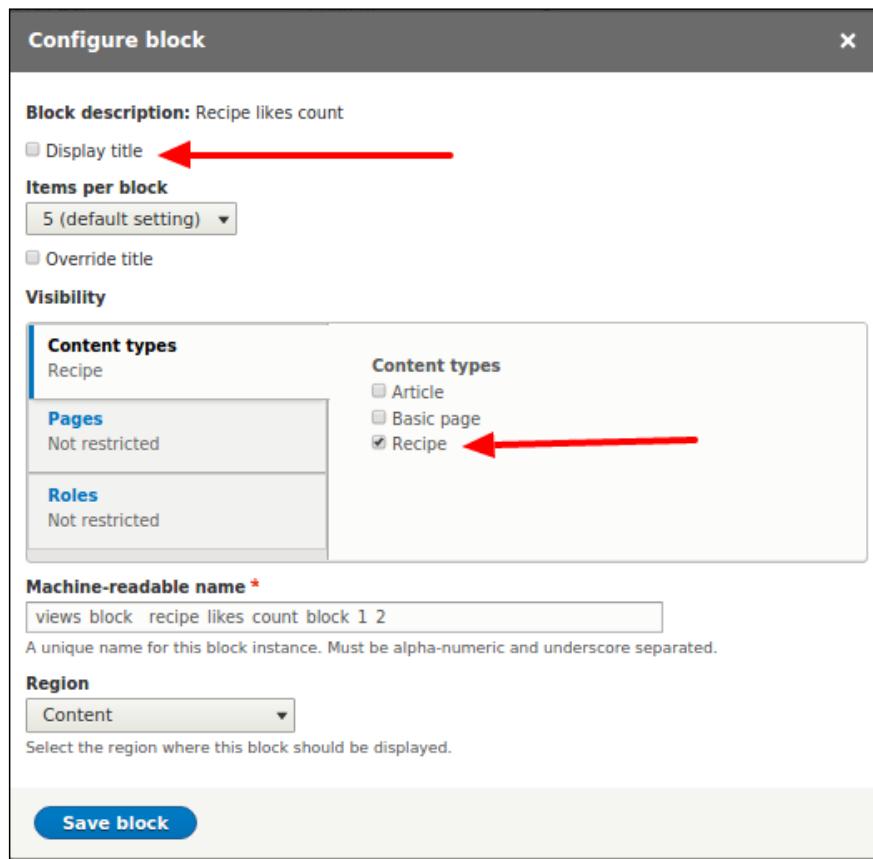


- 17.** Next, in the **FIELDS** section, click on the **Comment: Title** link. In the popup, click on the **Delete** link. In the same **FIELDS** section, click on the **Add** link. In the popup, search for **title** and select **Type** as **Content**. Check the **Title** checkbox and click on the **Apply(all displays)** button at the bottom.
- 18.** On the next screen, leave the default settings and click on the **Apply and continue** button.
- 19.** On the next screen, check the **Exclude from display** checkbox and click on the **Apply(all displays)** button at the bottom. Then, scroll down and click on the **Save** button to save the view settings.
- 20.** We are done creating a block view. Now we need to place this block in the Recipe content page. Click on the **Structure** link in the **Admin** toolbar, and then click on the **Block layout** link.

- 21.** Scroll down to the **Content** section and click on the **Place block** button. Next, in the popup, search for Recipe likes count and click on the **Place block** button for this:



- 22.** On the next screen, uncheck **Display title**. Under the **Content types** section check **Recipe** and leave the other settings as default. Click on the **Save block** button:



- 23.** Next, in the **Block layout** page, move the **Recipe likes count** block to above the **Main page content** block and scroll down to the bottom. Click on the **Save block** button:

| Region        | Type          | Block Name            | Status        | Action    |
|---------------|---------------|-----------------------|---------------|-----------|
| Breadcrumb    | System        | Breadcrumb            | Content       | Configure |
| Content       | core          | Page title            | Content       | Configure |
| Content       | core          | Tabs                  | Content       | Configure |
| Content       | Help          | Help                  | Content       | Configure |
| Content       | core          | Primary admin actions | Content       | Configure |
| Content       | Lists (Views) | Recipe likes count*   | Content       | Configure |
| Content       | System        | Main page content     | Content       | Configure |
| Sidebar first | Forms         | Search                | Sidebar first | Configure |
| Sidebar first | Menus         | Tools                 | Sidebar first | Configure |

- 24.** We are now done with all configurations. Now we need to check out the recipe content page to check how the number of likes will display on it. We will use the Devel module to generate dummy content and comments. Use Drush to download and enable the Devel module. Run the following commands in the d8dev root folder:

```
$ drush en devel -y
devel was not found.
[warning]
The following projects provide some or all of the extensions not
found:
[ok]
devel
Would you like to download them? (y/n): y
Project devel (8.x-1.x-dev) downloaded to /usr/share/nginx/html/
drupal-8.0.2//modules/devel.
[success]
Project devel contains 5 modules: webprofiler, kint,
devel_node_access, devel_generate, devel.
The following extensions will be enabled: devel
Do you really want to continue? (y/n): y
devel was enabled successfully.
[ok]
devel defines the following permissions: access devel
information, execute php code, switch users
```

```
$ drush en devel_generate -y
```

## *How Does It Taste? – Getting Feedback*

---

devel\_generate is already enabled.

[ok]

There were no extensions that could be enabled.

[ok]

- 25.** To generate dummy content, open up the d8dev site in your favorite browser, click on the **Configuration** link in the **Admin** toolbar, and then click on the **Generate content** link under the **DEVELOPMENT** section.
- 26.** On the next screen, check the **Recipe** content type, enter 10 in the **Maximum number of comments per node** field, and click on the **Generate** button. This will generate 50 Recipe contents with 10 comments each:

**Generate content** ★

Home » Administration » Configuration » Development

| CONTENT TYPE                               | COMMENTS              |
|--|-----------------------|
| Article                                    | No comment fields     |
| Basic page                                 | No comment fields     |
| <input checked="" type="checkbox"/> Recipe | • Recipe Review: Open |

Delete all content in these content types before generating new content.

How many nodes would you like to generate? \*

50

How far back in time should the nodes be dated?

1 week ago

Node creation dates will be distributed randomly from the current time, back to the selected time.

Maximum number of comments per node.

10

You must also enable comments for the content types you are generating. Note that some nodes will randomly receive zero comments. Some will receive the max.

Maximum number of words in titles \*

4

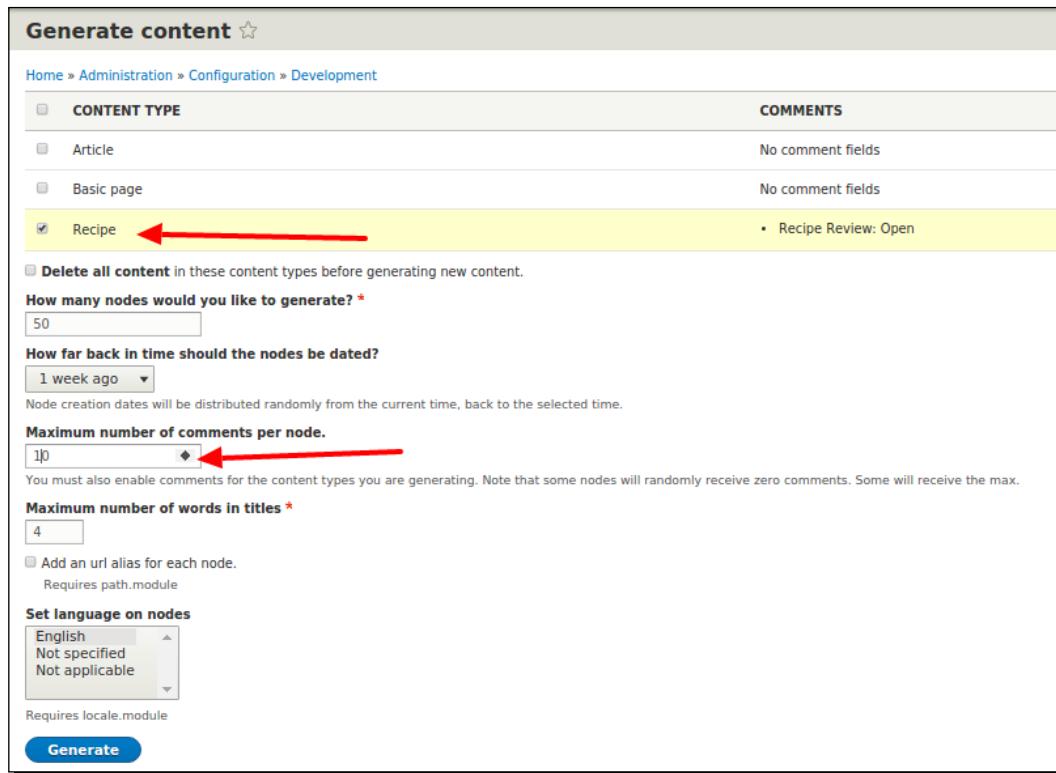
Add an url alias for each node.  
Requires path.module

Set language on nodes

English  
Not specified  
Not applicable

Requires locale.module

**Generate**



- 27.** Open any Recipe content that was just created by the devel module from the **Admin Content** page. We can see the count of likes as shown in the next screenshot. If we check all comments of this Recipe, we can find only 4(or how many Yes values) likes from the **Do you like it ?** field:

Aptent Meus

View Edit Delete Devel

4 likes ←

Submitted by Anonymous (not verified) on Wed, 01/01/2014 - 11:55

**Recipe images** Capto hos nostrud pla  
Decet et imputo jumer  
ludus mauris natu pro

- 28.** Only approved comments' likes will be counted, so if we get any comments, then the administrator has to update them as **Published comments**. To do this, we have to go to <http://localhost/admin/content/comment/approval>:

Unapproved comments ☆

Content Comments Files

Published comments Unapproved comments (0)

Home » Administration » Content » Comments

▼ UPDATE OPTIONS

Publish the selected comments Update

| SUBJECT | AUTHOR | POSTED IN | UPDATED | OPERATIONS |
|---------|--------|-----------|---------|------------|
|         |        |           |         |            |

### **What just happened?**

We created a **View** block and configured the comment settings to use for the Review and Like/Dislike recipes on our d8dev site. The process was straightforward and showed that sometimes adding a unique flare to a Drupal site doesn't take much development at all.

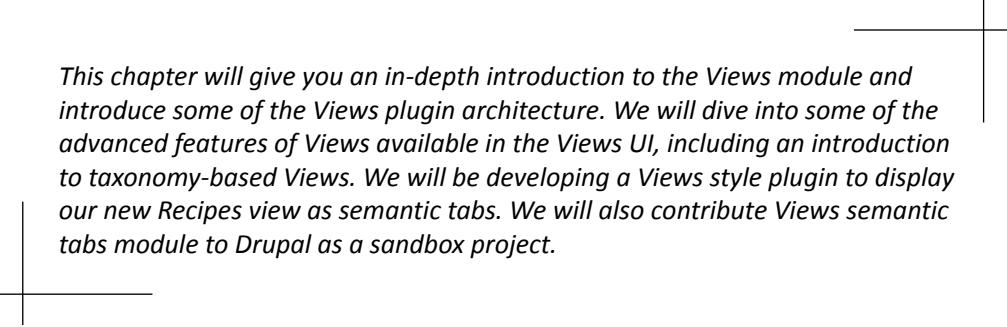
## **Summary**

In this chapter, we added some new features that will provide a way for visitors to interact with our d8dev site, and we enhanced some of those interactive features with comments and views. We also revisited the Colorbox File module that was introduced in the previous chapter, and made some modifications to it that allow more control over how captions are displayed in the Colorbox overlay.

In the next chapter, we will take a more in-depth look at the Views module. We will explore advanced configuration and developing a custom Views plugin that utilizes a custom jQuery plugin to display the Views output in semantic and progressive enhanced tabs.

# 9

## Advanced Views Development



*This chapter will give you an in-depth introduction to the Views module and introduce some of the Views plugin architecture. We will dive into some of the advanced features of Views available in the Views UI, including an introduction to taxonomy-based Views. We will be developing a Views style plugin to display our new Recipes view as semantic tabs. We will also contribute Views semantic tabs module to Drupal as a sandbox project.*

In this chapter, we will cover the following topics:

- ◆ Advanced Views configuration
- ◆ Views plugin development
- ◆ Creating a Drupal sandbox project

## **Views revisited – advanced configuration**

In the last few chapters, Custom Content Types and an Introduction to Module Development, we had a quick introduction to Views, and saw how easy it is to create a view with the new Views wizard's user interface. The new wizard-based creation for new views makes it easy to get started with Views, but does not include many of the more advanced Views configuration options. Even on the standard Views edit page, advanced configuration options are hidden away, so as not to overwhelm those who are new to Views. The beginning of this chapter will explore many of these advanced configuration options available with Views. Views configuration can get complex pretty quickly. So, in a way, advanced Views configuration is not any less complex than some of the PHP code we have written.

### **Random top rated recipe block**

The home page is still a bit plain and boring. We are going to use Views to create a block that will randomly showcase one of the top-rated recipes on the site. This will involve using Views filters and sort settings.

#### **Time for action – building a random top rated recipe block with Views**

We are going to go beyond the basic Views wizard view creation user interface and learn a few more advanced features and configuration of Views.

- 1.** Open our d8dev site in your browser, click on the **Structure** link in the **Admin** toolbar, and click on the **Views** link.
- 2.** Click on the **Add new view** link at the top of the **Views** page.
- 3.** Enter Random Top Rated Recipe as the **View name**.
- 4.** Select **Recipe** for the **of type** options.
- 5.** Check the **Create a block** checkbox.

Home > Administration > Structure > Views

**VIEW BASIC INFORMATION**

**View name \***  
Random Top Rated Recipe Machine name: random\_top\_rated\_recipe [Edit]

Description

**VIEW SETTINGS**  
Show: Content of type: Recipe sorted by: Newest first

**PAGE SETTINGS**  
 Create a page

**BLOCK SETTINGS**  
 Create a block  
Block title  
Random Top Rated Recipe

**BLOCK DISPLAY SETTINGS**  
Display format:  
Unformatted list of: fields

Items per block  
1|

Use a pager

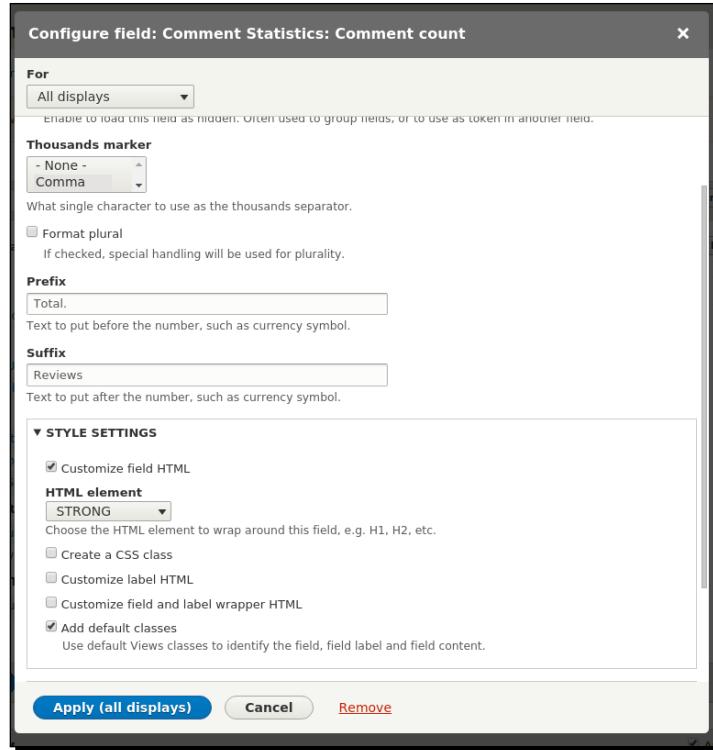
**Save and edit** **Cancel**

6. Next, click on the **Save and edit** button, as we want to configure some more advanced options that are not available as part of the basic block creation wizard. Now we are going to add the Recipe content fields that we want to display in this block. Remember, this block is going to be displayed on our d8dev site's front page, so we want to make it visually appealing. Note that the **Title** field is already included by default.

7. Click on the **Add** button for **FIELDS**, search for **image**, and select the checkbox for **Content: image** and click on the **Apply (all displays)** button. Note that Views shows you what node or content types the fields are associated with.

The screenshot shows the 'Add fields' dialog box. At the top, there is a search bar with 'image' typed into it and a dropdown menu labeled '- All -'. Below the search bar is a table with three columns: 'TITLE', 'CATEGORY', and 'DESCRIPTION'. There is one row in the table, which contains a checked checkbox, the title 'Image', the category 'Content', and the description 'Appears in: article, recipe.'. Below the table, the text 'Selected: Image' is displayed. At the bottom of the dialog box are two buttons: 'Add and configure fields' (highlighted with a blue border) and 'Cancel'.

8. For the **Configure field** settings for the image field, select **Medium (220\*220)** as the **Image style** and click on the **Apply (all displays)** button.
9. Click on the **Add** button for **FIELDS**, search for **Comment count**, and select the checkbox for **Comment Statistics: Comment count**. Then click on the **Apply(all displays)** button.
10. For the **Configure field** settings for **Comment count**, select **Total:** as the **Prefix** and **Reviews** as the **Suffix** value. In the **STYLE SETTINGS** section, check the **Customize field HTML** checkbox, select **STRONG** as the **HTML element** field, and click on the **Apply(all displays)** button.



11. Next, in the **SORT CRITERIA** section, click on the **Add** button, search for **Comment count**, and select the checkbox for **Comment Statistics: Comment count**. Then click on the **Apply(all displays)** button.
12. For the **Configure sort criteria** settings for **Comment count**, select **Sort descending** for the **Order** field and click on the **Apply(all displays)** button.



- 13.** Next, in the **SORT CRITERIA** section, click on the **Add** button, search for **Random**, and select the checkbox for **Random**. Then click on the **Apply(all displays)** button.
- 14.** In the next screen, click on the **Apply(all displays)** button. Then save the view and we can see a preview of this block as the following screenshot:



- 15.** Now your Views configuration should look similar to the following screenshot:

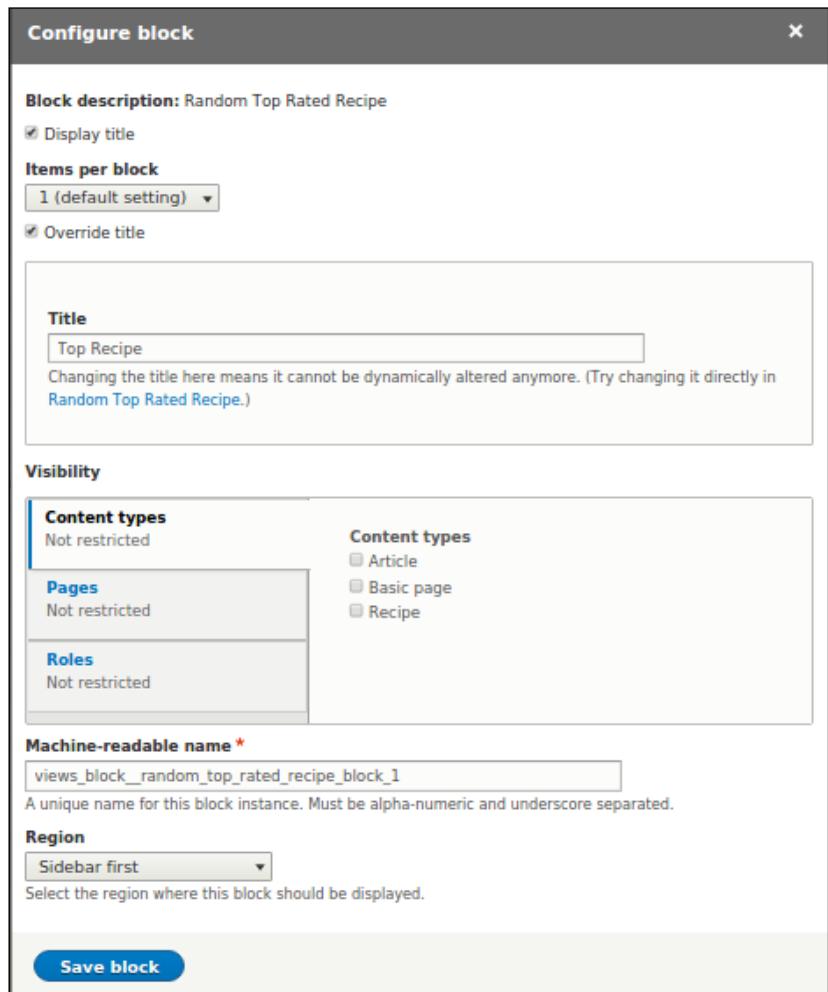
A screenshot of the Drupal 'Displays' configuration dialog for a 'comment count block'. The dialog is divided into several sections:

- TITLE**: Title: Random Top Rated Recipe
- FORMAT**: Format: Unformatted list | Settings. Show: Fields | Settings
- FIELDS**: Content: Title, Content: Image, Comment Statistics: Comment count
- FILTER CRITERIA**: Content: Publishing status (Yes), Content: Type (= Recipe)
- SORT CRITERIA**: Content: Authored on (desc)
- BLOCK SETTINGS**: Block name: comment count block, Block category: Lists (Views), Allow settings: Items per page, Access: Permission | View published content
- HEADER**: Add
- FOOTER**: Add
- NO RESULTS BEHAVIOR**: Add
- PAGER**: Use pager: Display a specified number of items | 1 item, More link: No, Link display: None

At the bottom, there are 'Save' and 'Cancel' buttons.

- 16.** Now we need to configure this new Views-based block to show up on the front page. Click on **Structure** in the **Admin** toolbar and click on the **Block Layout** link.
- 17.** Scroll down towards **Sidebar first region** and click on the **Place block** link. In the popup, search for **Random top** and click on the **Place block** button for the **Random Top Rated Recipe** block.

- 18.** In the next screen, check the **Override title** checkbox, enter **Top Recipe** as **Title**, and click on the **Save block** button.



## What just happened?

We used Views and leveraged some advanced configuration options to display the image and title of the top most recent recipes and also learned how easy it can be to make our d8dev site more interesting by adding a dynamic Views-based block.

## Taxonomy-based View with tabs

In this section, we are going to add another Views-based block to our front page. However, this will be a Taxonomy-based View instead of the Content-based Views that we have created so far. Taxonomy refers to the organization of information. As we learned in the previous chapter, taxonomy is a field-able entity. The Taxonomy module is a core module and it allows you to create vocabularies of terms to associate to other entity types so that they can be organized. So before that, we need to create a view with Taxonomy vocabulary with terms and associate those terms to our recipe content. We are going to add a vocabulary for organizing recipes by type of cuisine.

### Time for action – creating a cuisine vocabulary to organize recipes

Before we can create a Taxonomy-based view, we need to create a Drupal Taxonomy vocabulary:

1. Open our d8dev site in your browser, click on the **Structure** link in the **Admin** toolbar, and click on the **Taxonomy** link.
2. On the **Taxonomy configuration** page, click on the **Add vocabulary** link.
3. Enter **Type of Cuisine** as the **Name** input and click on the **Save** button. Now we will add some terms for our new vocabulary.
4. Click on the **Add term** link for our new **Type of Cuisine** vocabulary (make sure you are in the `admin/structure/taxonomy/manage/type_of_cuisine/overview` page):

Type of Cuisine ☆

List Edit Manage fields Manage form display Manage display

Home » Administration » Structure » Taxonomy » Cuisine

You can reorganize the terms in *Cuisine* using their drag-and-drop handles, and group terms under a parent term by sliding them under and to the right of the parent.

+ Add term

NAME WEIGHT OPERATIONS

No terms available. [Add term](#).

Show row weights

5. Enter American as **Name** of our first term and click on the **Save** button.
6. Repeat the process and add the terms Asian and Thai. Now we are going to add a taxonomy field to our Recipe content type.
7. Click on the **Structure** link in the **Admin** toolbar, click on the link for **Content types**, and click on the **Manage fields** link for our Recipe content type.
8. Click on the **Add field** link. In the next screen, select **Taxonomy term** as **Add a new field** and enter *recipeCuisine* as **Label**. Click on the **Save and continue** button.

Add field ☆

Home » Administration » Structure » Content types » Recipe » Manage fields

Add a new field Re-use an existing field

Taxonomy term or - Select an existing field -

**Label \***  
recipeCuisine Machine name: field\_recipecuisine [Edit]

Save and continue

- 9.** In the next screen, select **Unlimited** as **Allowed number of values** and click on the **Save field settings** button. In the next screen, check **Type of Cuisine** under **REFERENCE TYPE | Vocabularies** and click on the **Save settings** button.

**recipeCuisine settings for Recipe ☆**

[Edit](#) [Field settings](#)

Home » Administration » Structure » Content types » Recipe » Manage fields

✓ Updated field *recipeCuisine* field settings.

**Label \***  
recipeCuisine

**Help text**

Instructions to present to the user below this field on the editing form.  
Allowed HTML tags: <a> <b> <big> <code> <del> <em> <i> <ins> <pre> <q> <small> <span> <strong> <sub> <sup> <tt> <ol> <ul> <li> <p> <br> <img>  
This field supports tokens.

Required field

**▼ DEFAULT VALUE**  
The default value for this field, used when creating new content.  
**recipeCuisine**

**▼ REFERENCE TYPE**

**Reference method \***  
Default

**Vocabularies \***

Type of Cuisine (arrow)

Tags

Create referenced entities if they don't already exist

[Save settings](#) [Delete](#)

- 10.** Click on the **Content** link in the **Admin** bar, then click on the edit link for one of the recipes, scroll down to our new **recipeCuisine** field, select American, and click on the **Save** button. Repeat the process for selecting Asian for the Cashew Chicken with Edamame, and Thai and Asian for the other recipes.

## **What just happened?**

We got a quick introduction to Drupal Taxonomies and created a vocabulary to organize the d8dev recipes by the types of cuisine.

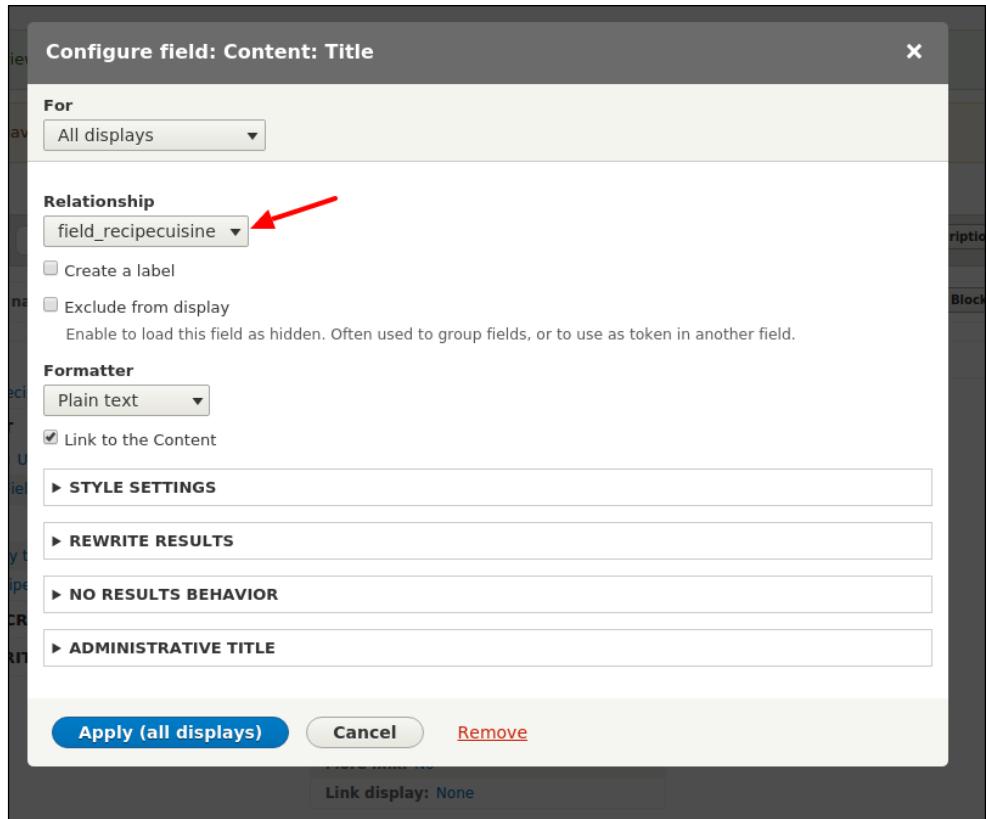
Now that we have added a new vocabulary for associating Recipe content to the types of cuisine, we are ready to use this in a new Views-based block. We are going to create a Views-based block that displays our d8dev site's newest recipe entries by cuisine type. In addition to that, we are going to sort the list of recipes by the cuisine type with the least number of associated recipes. This will help us to promote cuisine types with fewer recipes. Finally, we want a tab-based user interface with a tab for each cuisine type, and the contents of that tab to be the five most recent recipes for that cuisine type. Let's go through step by step.

## **Time for action – creating a Recipes by cuisine type Views block**

We have created a new vocabulary and associated it to our Recipe content type. Now we will learn how to use a custom vocabulary with a view.

1. Click on the **Structure** link in the **Admin** toolbar, click on the **Views** link, and click on the **Add new view** button.
2. On the next page, enter **Recipes by Cuisine** as the **View** name, select **Taxonomy terms** for the **Show** select list, and select **Type of Cuisine** for the **Type** select list.
3. Check the **Create a block** checkbox under the **Block settings** section.
4. Leave the remaining default settings as they are and click on the **Save and edit** button.
5. Views automatically adds the Taxonomy term **Name** field, but we also want to display the most recent recipes associated to each of those cuisine terms. However, if you click on the **Add** button for **FIELDS**, you will note that there is no **Content** field available. We will use the Views **RELATIONSHIPS** configuration to add a relationship between the Recipe content and the taxonomy terms we are showing.
6. Click on the **Add** button for **RELATIONSHIPS**, check the **Content using field\_recipescuisine** relationship, and then click on the **Apply(all displays)** button.
7. On the next screen, leave the remaining default settings as they are and click on the **Apply(all displays)** button.
8. Now click on the **FIELDS add** button; there are content fields available. Type **title** into the **Search** input, select it, and click on the **Apply(all displays)** button.

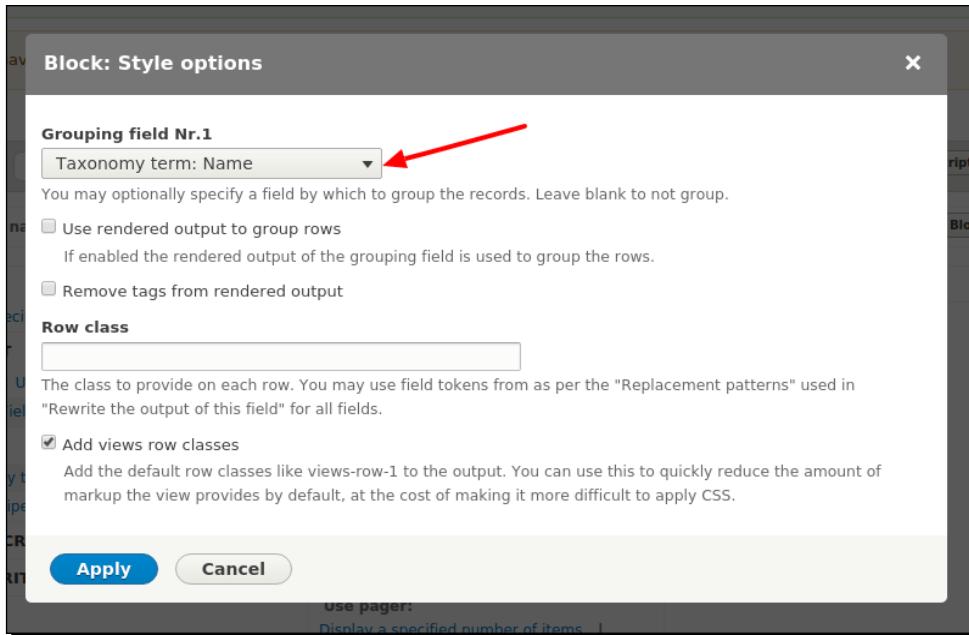
Note that there is a **Relationship** select list for the field configuration. All the content fields on a taxonomy terms-based view require a relationship. So this will default to the first relationship listed.



9. Leave the **Create a label** checkbox unchecked, as we only want to display the title itself. We will leave the **Link to this Content...** checkbox checked, so that the users have the ability to navigate to the full recipe. Click on the **Apply (all displays)** button.

Now if you scroll down to the bottom of the Views configuration page, you will see a preview of this Views output and you will see that we are displaying cuisine type term names and recipe titles, but we want to group the recipe titles by term names.

- 10.** Next, under the **FORMAT** section, click on the **Settings** link for Format and then select **Taxonomy term: Name** for the **Grouping field Nr.1** field and click on the **Apply** button



- 11.** You will see a preview of this View's output and you will see that we are displaying cuisine type term names and recipe titles, as we grouped the recipe titles by term names and term names were displaying twice. So we will hide them in the **FIELDS** settings.
- 12.** Now click on the **Taxonomy term: Name** link and check the **Exclude from display** checkbox, as we only want to display the content titles. Then click on the **Apply (all displays)** button.
- 13.** Now we will add a sort criterion to display the grouped terms with the most recent recipes first. Click on the **Add** button for **SORT CRITERIA**, type **Authored on** in the **Search** field, select the **Authored on** field, and then click on the **Apply (all displays)** button.

- 14.** On the next screen, select **Sort descending** for the **Order** field and click on the **Apply (all displays)** button. The preview for this View should now look similar to the following screenshot. Note: the following content is Devel module generated content.

| Content                       |
|-------------------------------|
| <b>Asian</b>                  |
| Acsi                          |
| Camur Luctus Populus Proprius |
| Volutpat                      |
| Facilisis Loquor Ullamcorper  |
| Vicis                         |
| <b>American</b>               |
| Quibus                        |
| Lucidus Magna Pneum           |
| Abdo Cogo Iriure Vel          |
| <b>Thai</b>                   |
| Thai chicken curry            |

- 15.** Next, click on the **Save** button for this View.

### **What just happened?**

We created a Views-based block of recipes displayed by the cuisine type name. Although everything appears fine on the surface, there is a problem with the groupings and the limits for our new view. We wanted to display three different cuisines and five recipes per cuisine type, but the view we created is only limiting the total number of rows being returned. If we were to add one more recipe, then that recipe would be displayed. However, the sixth-oldest recipe would drop off, and if the newly added recipe happened to be of type Thai or Asian, then the American grouping would disappear. So, we would only be left with two groups of cuisine types. It turns out that this is a rather complex problem to solve with SQL, but there is a contrib module that will allow us to get the exact results that we want.

The Views Field View module ([http://drupal.org/project/views\\_field\\_view](http://drupal.org/project/views_field_view)) enables a Global Views field that allows you to embed another View as a field of a parent view, sort of like a set of Russian dolls. For actual production use, however, do note that there are some pretty serious performance implications for using this approach, as there will be a total of four SQL queries instead of one. So you definitely want to make sure you understand Views caching and Drupal caching in general before you use an approach like this on a production site.

## Time for action – installing and using the Views Field View module for our Recipe by Cuisine Type View

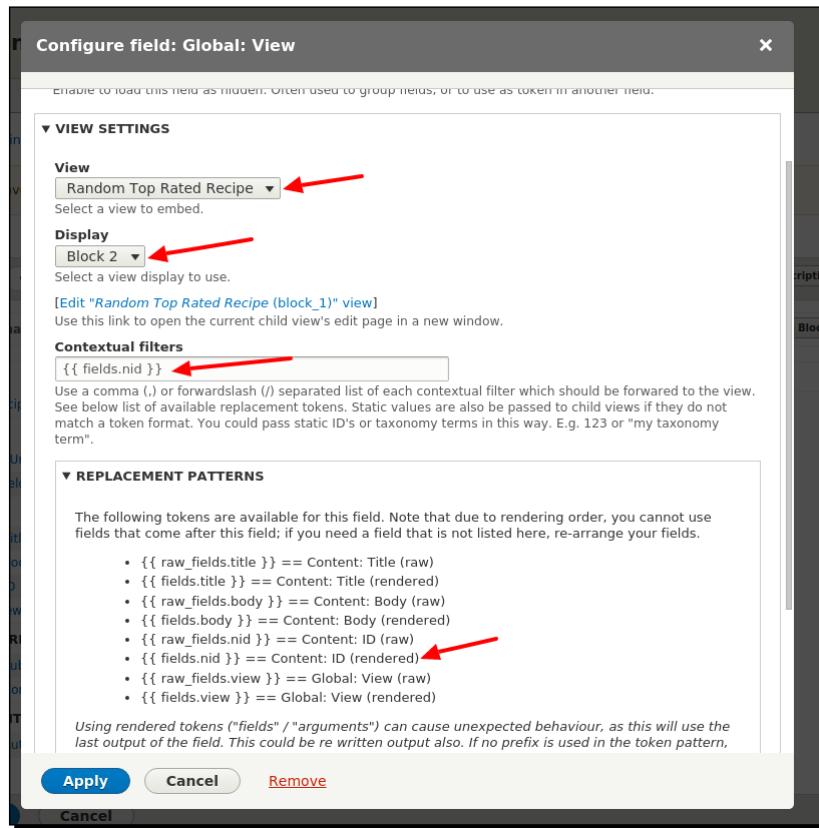
By installing and using the Views Field View module, we are going to learn how there are a number of Views related contrib modules that extend the features and capabilities of the Views module.

- 1.** First, we need to install the View Field View module. Open the Terminal (Mac OS X) or Command Prompt (Windows) application and go to the root directory of our d8dev site.
- 2.** Use Drush to download and enable the Views Field View module:

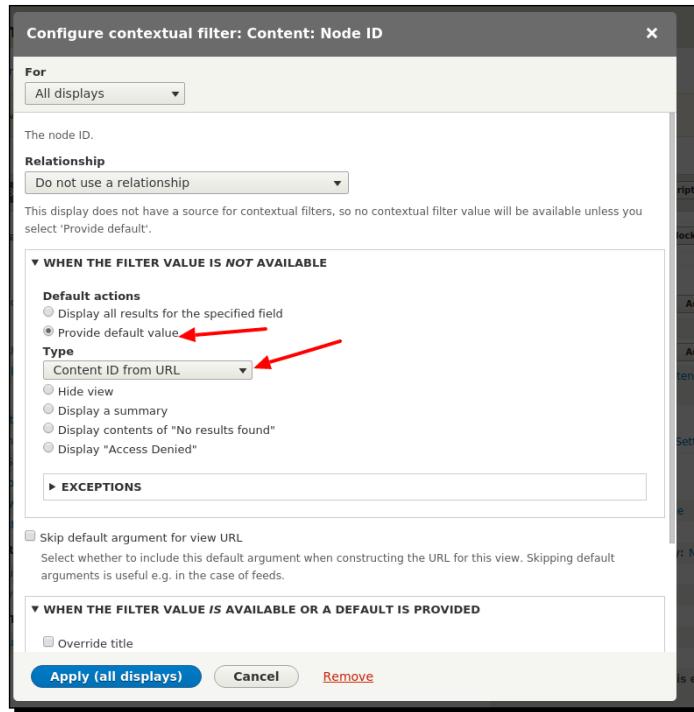
```
$ drush dl views_field_vies
Project views_field_view (8.x-1.0-beta2) downloaded to /var/www/
html/d8dev/modules/views_field_view. [success]
$ drush en views_field_view
The following extensions will be enabled: views_field_view
Do you really want to continue? (y/n): y
views_field_view was enabled successfully. [ok]
```

- 3.** Now, before we modify our Recipes by Cuisine view, you need to understand how the Views Field View functionality works. We will remove the recipe title field and add a **Global: View** field. The **Global: View** field allows us to specify another view to use as the contents of the field, instead of a field on our Recipe content type. It allows us to pass any other field available for our view as an argument to pass as a contextual filter to the other view being used as the contents of the field. I know it sounds pretty complicated, and that is why we are going to walk through it together, nice and slow. To start with, we need to create a view to use as the Views Field View, where it will list out the recipes ordered by post date in descending order, so we will use the Views Field View field to display the contents of our Recipe List view inside the rows of our Recipes by Cuisine view.
- 4.** Click on the **Structure** link in the **Admin** toolbar, then click on the **Views** link, and click on the **Edit** button for our random top rated recipe View.

5. At the top of the next page, click on the **Add** button and then click on the **Block** link.
6. Click on the **Add Field** link and search for node id in the input search box. Check the **Exclude from display** checkbox and click on the **Apply (this displays)** button.
7. Again click on the **Add Field** link and search for view in the input search box. On the next screen, in the **VIEW SETTINGS** section, select **Recipe likes count** as **View**, **Block** as **Display**, and `{{ fields.nid }}` as **Contextual filters**. Click on the **Apply (this displays)** button.



8. Now, in the **ADVANCED** section, click on the **Add** link for **Contextual filters**. In the popup, search for the node ID keywords. Now check the **Node ID** checkbox and click on the **Apply (this displays)** button. In the next screen, leave the settings as default and click on the **Apply and continue** button. Note: make sure you selected **This block (override)** at the top.
9. On the next screen, under **WHEN THE FILTER VALUE IS NOT AVAILABLE** section, select the **Provide default value** radio button. Select **Type** as **Content ID from URL** and click on the **Apply(this displays)** button at the bottom.

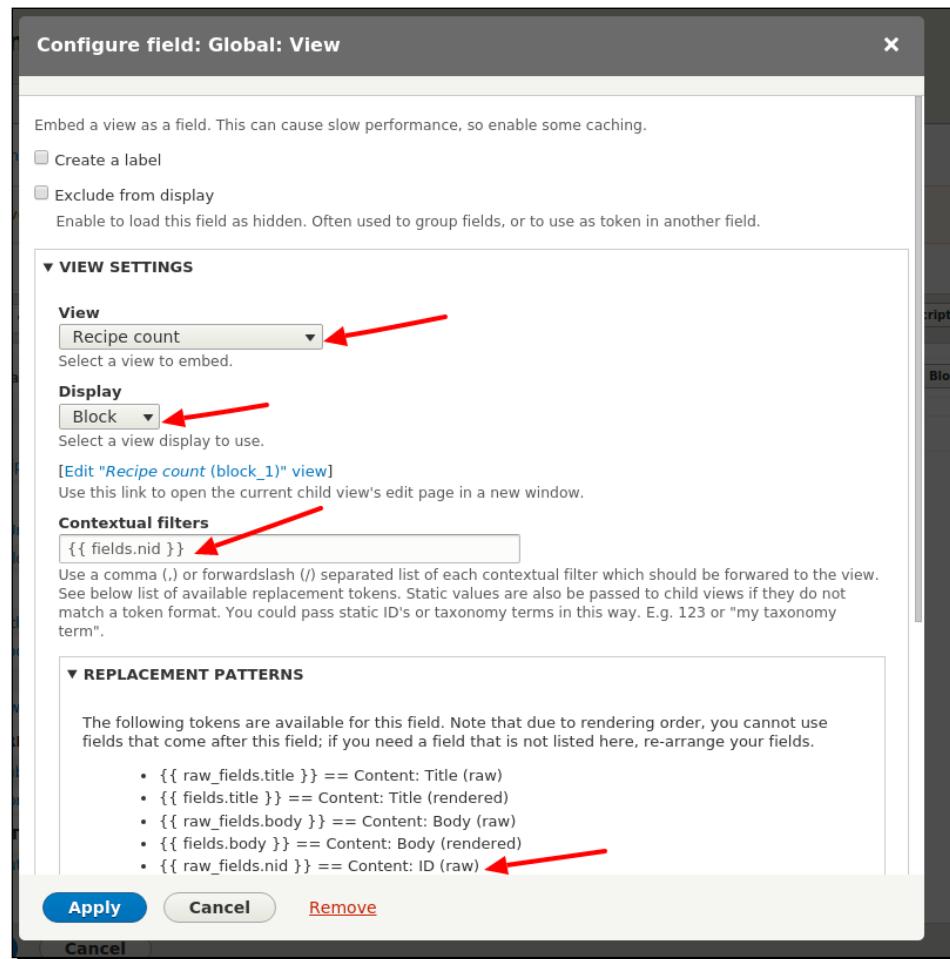


- 10.** Then save the view and we can see a preview of this block as per the following screenshot:



- 11.** Next, click on the **Structure** link in the **Admin** toolbar, then click on the **Views** link, and click on the Edit button for our Recipes by Cuisine View.
- 12.** At the top of the next page, click on the **Add** button and then click on the **Block** link.
- 13.** Click on the **Add Field** link and search for node ID in the input search box. Check the **Exclude from display** checkbox and click on the **Apply (this displays)** button.

- 14.** Again click on the **Add Field** link and search for view in the input search box. On the next screen, in the **VIEW SETTINGS** section, select **Random Top Rated Recipe** as **View**, **Block 2** as **Display**, and `{{ raw_fields.nid }}` as **Contextual filters**. Click on the **Apply (this displays)** button.



- 15.** Then save the view and we can see preview of this block as per the following screenshot:



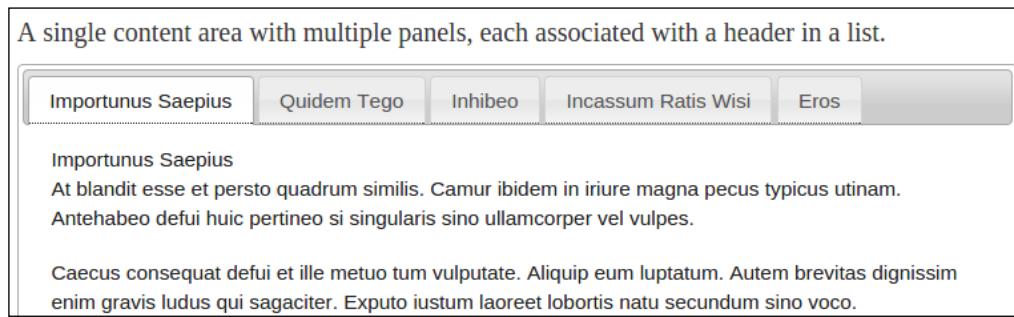
- 16.** Now we need to configure this new Views-based block to show up on the front page. Click on **Structure** in the **Admin** toolbar and click on the **Block Layout** link.  
**17.** Scroll down towards **Sidebar first Region** and click on the **Place block** link. In the popup, search for **Recipes by** and click on the **Place block** button for the **Recipes by Cuisine** block.

### ***What just happened?***

With the help of the Views Field View module, we created a view that displays our recipe content.

## Tabbed Views display

In order to make the Recipes by Cuisine and to make the block more visually appealing, and appear more organized in the viewable area of our d8dev front page, we are going to display each cuisine type as a tab, and have recipes for the active tab. We are going to use a JavaScript-based approach for displaying our groups of recipes by cuisine in a tabbed interface. Take a look at the jQuery UI tabs page (<http://jqueryui.com/tabs/>) and you will see an example of how we can display Recipe by Cuisine in a tab.



The reason I am pointing out jQuery based UI tabs is because Drupal 8 includes the JavaScript library. So, it makes a lot of sense to use a JavaScript widget for the tab that is already available to us as part of the core Drupal 8 install. However, the markup that is currently being generated for our Recipe by Cuisine View will be fairly difficult to integrate with the jQuery UI tabs, because jQuery UI tabs are intended to handle the tabs and the tab content in separate HTML containers. Take a look at the example markup from the jQuery UI tabs page in the previous screenshot to see what I mean.

```
<div id="tabs">
  <ul>
    <li><a href="#tabs-1">Tab1 title</a></li>
    <li><a href="#tabs-2">Tab2 title</a></li>
    <li><a href="#tabs-3">Tab3 title</a></li>
  </ul>
  <div id="tabs-1">
    <p>Tab1 content</p>
  </div>
  <div id="tabs-2">
    <p>Tab2 content</p>
  </div>
```

---

```
<div id="tabs-3">
  <p>Tab3 content</p>
</div>
</div>
```

The markup that Views is generating for our Recipes by Cuisine is more semantic. It keeps the group titles with the associated content (Views actually generates a lot more markup than this, so take a look at the source output for our Recipes by Cuisine View in your browser). Basically, a simplified version of what Views generates for the default format of HTML list is closer to the following:

```
<div>
  <ul>
    <li>content 1</li>
    <li>content 2</li>
  </ul>
</div>
```

Therefore, if we want to use jQuery UI tabs then we have to modify the markup that Views is generating for our Recipes by Cuisine View. A Views style plugin would allow us to generate exactly the type of markup that is typically used with jQuery UI tabs. However, since we are going to write a custom plugin for Views anyways, why write one for creating tabs that isn't very semantic and is limited in regards to progressive enhancement?

## Time for action – developing a Views style plugin for semantic tabs

We are going to create a new module for our introduction to plugins for Views. And we will contribute it to Drupal in the next topic.

1. Open PhpStorm and navigate to the /modules/custom folder in our d8dev project.
2. Create a new folder named `views_semantic_tabs`—the name of our new module.
3. Create a new file named `views_semantic_tabs.info.yml` and enter the following information:

```
name: Views semantic tabs
type: module
description: Provides a views style plugin to display views results in jQuery UI Tabs.
core: 8.x
package: Views
dependencies:
  - views
```

In Drupal 7, we use the `hook_views_plugins()` hook function to register new plugins. Drupal 8, on the other hand, depends on annotations and auto-loading to discover any plugins such as blocks and views styles. The auto-loading concept allows us to put a plugin file in a predefined directory, and Views/Blocks find it as needed. The metadata of any plugin specified inside the plugin's file using comment block, is called as annotations.

4. To find our style plugin by views, we have to place it in the `src\Plugin\views\style` folder which is inside our custom module `modules/views_semantic_tabs` directory. So here we wish to build semantic tabs style plugin that displays jQuery UI tabs based style.
5. Create a new file `ViewsSemanticTabs.php` inside the `/views_semantic_tabs/src\Plugin\views\style` directory. The skeleton of the semantic tabs style plugin looks like the following:

```
<?php

/**
 * @file
 * Contains \Drupal\views\Plugin\views\style\HtmlList.
 */

namespace Drupal\views_semantic_tabs\Plugin\views\style;

use Drupal\Core\Form\FormStateInterface;
use Drupal\views\Plugin\views\style\StylePluginBase;

/**
 * Style plugin to render each item in an ordered or unordered
 * list.
 *
 * @ViewsStyle(
 *   id = "views_semantic_tabs",
 *   title = @Translation("Semantic tabs"),
 *   help = @Translation("Configurable semantic tabs for views
 * fields."),
 *   theme = "views_semantic_tabs_format",
 *   display_types = {"normal"}
 * )
 */
class ViewsSemanticTabs extends StylePluginBase {

  /**
   * Does the style plugin allows to use style plugins.
}
```

```

*
* @var bool
*/
protected $usesRowPlugin = TRUE;

/**
 * Does the style plugin support custom css class for the
rows.
*
* @var bool
*/
protected $usesRowClass = TRUE;

/**
 * Does the style plugin support grouping of rows.
*
* @var bool
*/
protected $usesGrouping = FALSE;

/**
 * Set default options
*/
protected function defineOptions() {
    $options = parent::defineOptions();
    $options['group'] = array('default' => array());
    return $options;
}

/**
 * Render the given style.
*/
public function buildOptionsForm(&$form, FormStateInterface
$form_state) {
    parent::buildOptionsForm($form, $form_state);
    $options = array('' => $this->t('- None -'));
    $field_labels = $this->displayHandler
->getFieldLabels(TRUE);
    $options += $field_labels;
    $grouping = $this->options['group'];
    $form['group'] = array(
        '#type' => 'select',
        '#title' => $this->t('Grouping field'),
        '#options' => $options,
}

```

```
        '#default_value' => $grouping,
        '#description' => $this->t('You should specify a
field by which to group the records.'),
        '#required' => TRUE,
    );
}
}
```

Our class `ViewsSemanticTabs` needs to inherit from the `StylePluginBase` class, and therefore we are using a keyword. Also, we are using `FormStateInterface` to user forms inside our class. This file is in the correct location and we have an annotation above the class definition. This annotation was giving the ID of the style, theme, and display types. So views will find this plugin and it will be available for selection in the views format style settings. The protected `$usesRowPlugin` property is necessary to this plugin which lets us select whether we would like to display fields or rendered content in the view display. The protected `$usesRowClass` property also required for our case which does the style plugin support custom CSS class for the rows. The protected `defineOptions()` method is used to define default options which will show in the settings form. And the protected `buildOptionsForm()` method is used to define any custom form values which will show in the settings form. `parent::buildOptionsForm($form,` `$form_state)`; allows us to access the output of the class we are extending and manipulate that output. We are making the `$form['group']` field required since the views HTML render output requires a grouping field name.

6. The output of views will live in the template file and the template file will need to be placed inside the `templates` directory with `views-semantic-tabs-format.html.twig` as the filename inside our module. We don't need to implement the `hook_theme()` hook function because it will be automatically registered based on the specified theme name in the annotation `views_semantic_tabs_format`. The template filename is derived from the theme name by replacing the underscores with hyphens and followed by the `html.twig` extension.

```
<div class="views-semantic-tabs">
    <ul>
        {%
            set i = 1
        %}
        {%
            for row in rows.group
        %}
            <li><a href="#tabs-{{ i }}">{{ row }}</a></li>
        {%
            set i = i + 1
        %}
        {%
            endfor
        %}
    </ul>
    {%
        set j = 1
    %}
    {%
        for row in rows
    %}
        <div id="tabs-{{ j }}">{{ row.content }}</div>
```

```

    {%
      set j = j + 1
    %}
  {% endfor %}
</div>

```

First, we need to understand that this template will be used for each tab. We had to wrap the group field values as a separate set which are part of `<li>` tags in `<ul>`. We are wrapping rows data which is the content for the tab and wrapped with the `div` tag with unique `tabs-id` attributes.

7. The variables in the template is provided by views. Preparing variables for a twig file can be done with the `template_preprocess_views_semantic_tabs_format()` function in the `.module` file. This function name is defined based on the specified name in the annotation `views_semantic_tabs_format`. Create a new file `views_semantic_tabs.module` inside the `/modules/views_semantic_tabs` directory. The skeleton of the semantic tabs style plugin looks like the following:

```

<?php

/**
 * @file
 * Contains views_semantic_tabs.module..
 */

use Drupal\Core\Routing\RouteMatchInterface;
use Drupal\Core\Template\Attribute;

/**
 * Implements hook_help().
 */
function views_semantic_tabs_help($route_name,
RouteMatchInterface $route_match) {
  switch ($route_name) {
    // Main module help for the views_semantic_tabs module.
    case 'help.page.views_semantic_tabs':
      $output = '';
      $output .= '<h3>' . t('About') . '</h3>';
      $output .= '<p>' . t('Provides a views style plugin to
display views results in jQuery UI Tabs.') . '</p>';
      return $output;

    default:
  }
}

/**

```

```
* Prepares variables for Views HTML list templates.
*
* @param array $variables
*   An associative array containing:
*   - view: A View object.
*/
function template_preprocess_views_semantic_tabs_format(&$variables) {
    $handler = $variables['view']->style_plugin;
    $view = $variables['view'];
    $rows = $variables['rows'];
    $style = $view->style_plugin;
    $fields = &$view->field;
    $options = $style->options;
    $variables['fields'] = $fields;

    // add jquery & jquery ui
    $variables['view']->element['#attached']['library'][] =
      'core/jquery.ui.tabs';
    $variables['view']->element['#attached']['library'][] =
      'core/jquery';
    $variables['view']->element['#attached']['library'][] =
      'views_semantic_tabs/views-semantic-tabs';

    // Get first group field selected
    if($options) {
        $first_group_field = $options['group'];
    }

    // Add tabs id to main div
    $variables['attributes'] = new Attribute(array('id' =>
      'tabs'));
    $fields = &$view->field;

    $variables['default_row_class'] =
      !empty($options['default_row_class']);
    foreach ($rows as $id => $row) {
        // todo : group field value should be plain text
        $field_output = $handler->getField($id,
          $first_group_field);
        $variables['rows'][$row]['group'][] = $field_output;
        $variables['rows'][$id] = array();
        $variables['rows'][$id][$content] = $row;
        $variables['rows'][$id]['attributes'] = new
        Attribute();
```

```

        if ($row_class = $view->style_plugin->getRowClass($id)) {
            $variables['rows'][$id]['attributes']
                ->addClass($row_class);
        }
    }
}

```

There are two functions, `template_preprocess_views_semantic_tabs_format()` and `hook_help()`. At the very top of our `template_preprocess_views_semantic_tabs_format()` function, we are getting the required handler, rows, fields, and style properties from the `$variables` variable. Then we are attaching `jquery.ui.tabs`, jQuery core jQuery libraries to the `$variable`, because it requires both jQuery libraries. And also we are attaching the custom `views-semantic-tabs` library which has custom JS file where we will define jQuery code to work tabs. In the next lines, we are building rows data which is a part of tab content. And we declared the help function to define simple documentation about this module. For more information about the `hook_help()` function, go to [https://api.drupal.org/api/drupal/core!modules!help!help.api.php/function/hook\\_help/8](https://api.drupal.org/api/drupal/core!modules!help!help.api.php/function/hook_help/8).

8. Next, right-click on the new theme folder, create a new file named `views_semantic_tabs.libraries.yml`, and add the following code to that file:

```

views-semantic-tabs:
  version: VERSION
  js:
    js/views-semantic-tabs.js: {}
  dependencies:
    - core/jquery
    - core/drupal

```

This is the library file which we attached in the `template_preprocess_views_semantic_tabs_format()` function in the last step. This defines the `views-semantic-tabs.js` file which has tabs jQuery code to work jQuery tabs as per the rendered HTML structure defined in the twig file.

9. Next, right-click on the module folder and create a new folder named `js`. Again, right click on the `js` folder and create a file named `views-semantic-tabs.js`. Add the following code to that file:

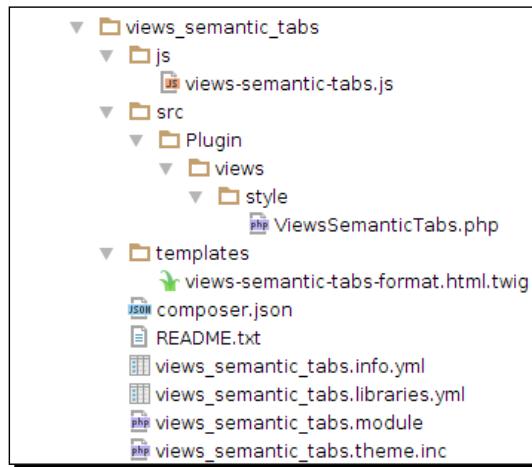
```

(function ($) {
  $(function() {
    $('.views-semantic-tabs').tabs();
  });
})(jQuery);

```

Note how the entire block of JavaScript code is wrapped with `(function ($) { ... })(jQuery);`. This is a new JavaScript namespacing feature of Drupal 7 and Drupal 8, and allows other JavaScript libraries to be used with Drupal with less likelihood of conflicts. `$(".views-semantic-tabs").tabs();` is the code which makes our twig rendered HTML code work as tabs.

- 10.** Now that we have completed all the code, our `views_semantic_tabs` folder should look similar to the following screenshot:



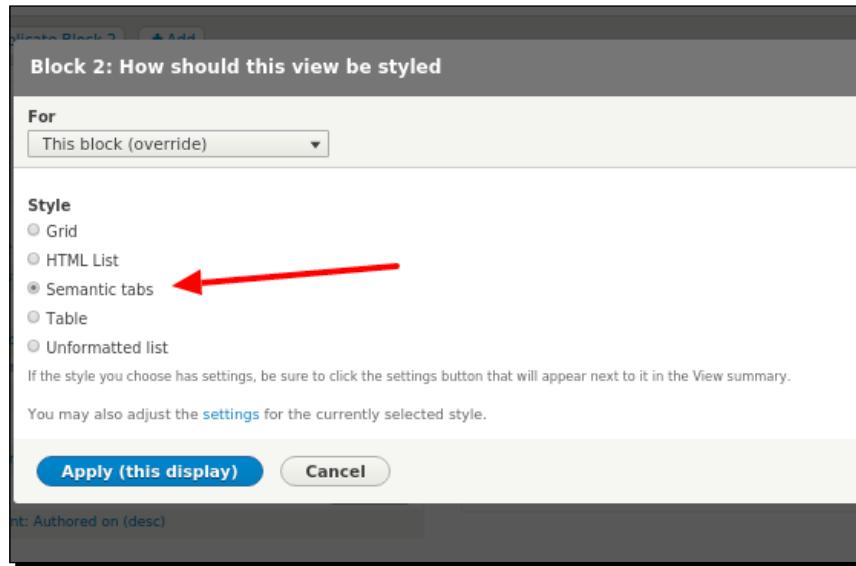
- 11.** Now we are ready to test our new Views style plugin by applying it to our Recipes by Cuisine view, but first we need to enable our new module. We could use Drush for this, but I would like to enable custom modules in the browser so that I can see my new module.
- 12.** Open our d8dev site in your browser, click on the **Extend** link in the **Admin** toolbar, and scroll down to the **Views** section of the modules or search for `views semantic` in the search input box.
- 13.** You should see our new Views Semantic Tabs module listed along with the other Views modules that we had installed.

The screenshot shows the Extend page with the title 'Extend' and a star icon. There are two buttons at the top: 'List' and 'Uninstall'. Below them is the breadcrumb navigation 'Home > Administration'. A note says 'Download additional contributed modules to extend your site's functionality.' Another note says 'Regularly review available updates to maintain a secure and current site. Always run the update script each time a module is updated. Enable the Update Manager module to update and install modules and themes.' A search bar contains the text 'views semantic' with a red arrow pointing to it. Below the search bar is a placeholder text 'Enter a part of the module name or description'. Under the heading '▼ VIEWS' is a module card for 'Views semantic tabs'. It has a checked checkbox, a red arrow pointing to it, and the text 'Provides a views style plugin to display views results in jQuery UI Tabs.' At the bottom of the card is a blue 'Install' button.

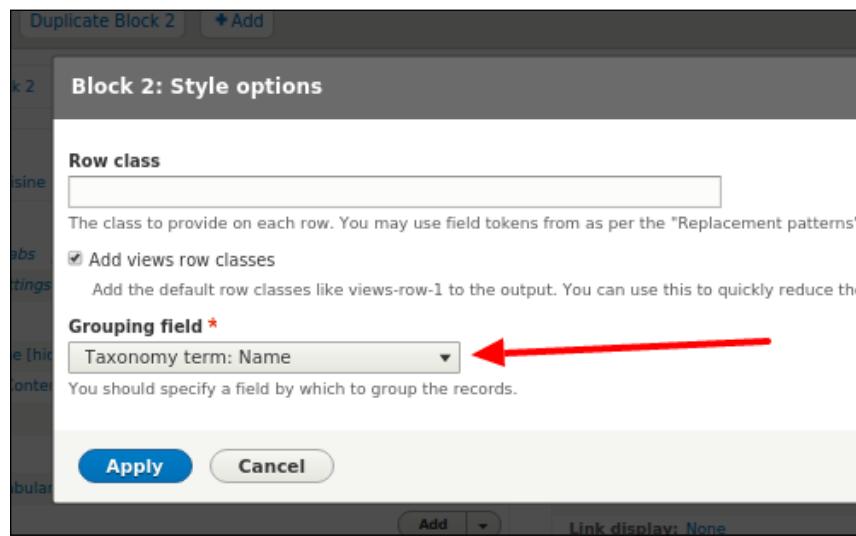
- 14.** Check the checkbox to enable our new module and click on the **Install** button.
- 15.** Next, click on the **Home** link in the **Admin** toolbar, then click on the **Contextual links** button for our Recipes by Cuisine view, and click on the **Edit view** link.

The screenshot shows the 'Recipe by Cuisine' view. At the top, there is a navigation bar with three tabs: 'Thai', 'Asian', and 'American'. Below the navigation bar, the title 'Chilli Chicken 65 indian style' is displayed. An image of the dish is shown, which is a plate of spicy chicken pieces garnished with coriander. At the bottom of the view, the text 'Total:3Reviews' is visible.

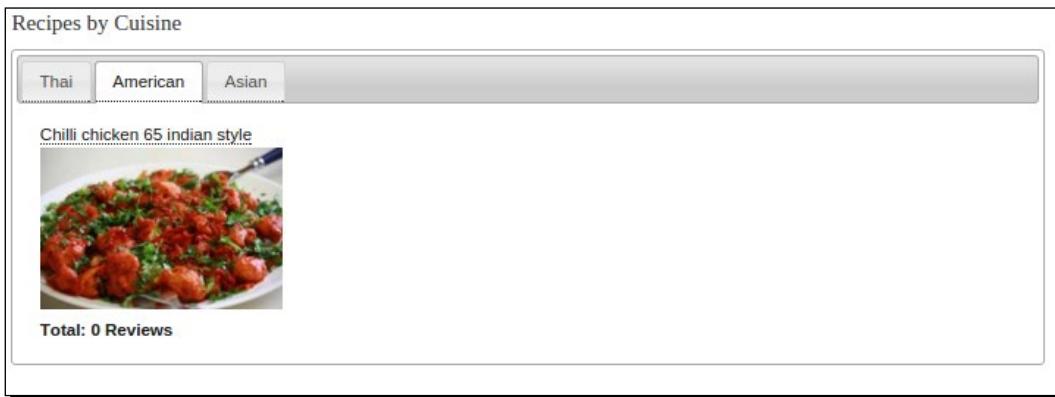
- 16.** Under **FORMAT**, click on the **Unformatted list** link. The view style settings form now includes our new Views style plugin: **Semantic tabs**.



- 17.** Select our **Semantic tabs** style and click on the **Apply (this display)** button.
- 18.** Next, on the **Style options** screen, select **Taxonomy term: Name** as **Grouping field** and click on the **Apply** button. Note that **Grouping field** is required as we specified in our plugin class.



**19.** Now, click on the **Save** button to save our changes to the view. You should now have a Recipe by Cuisine view block that looks similar to the following screenshot:



### ***What just happened?***

Although this example was fairly complex, the actual code is pretty straightforward once you get your head wrapped around some of the development concepts for Views plugins. We were able to create a custom Views style plugin that will enhance the display of content on our d8dev site.

## **Time for another Recipe**

Here is a little bit of spicy Americana for you—Kurt's Classic Chili. Add it to the d8dev site and checkout the Recipes by Cuisine view from the previous section (the secret ingredient is the bay leaves).



- ◆ **name:** Kurt's Classic Chili
- ◆ **description:** There is nothing like a warm bowl of chili on a cold winter day. The homemade chili powder really gives this dish a distinct and delicious flavor.
- ◆ **recipeYield:** Eight servings
- ◆ **prepTime:** 30 minutes
- ◆ **cookTime:** 60 minutes
- ◆ **ingredients:**
  - ❑ One pound of ground beef
  - ❑ Two tablespoons of olive oil
  - ❑ One large sweet onion, chopped
  - ❑ Six cloves garlic, crushed
  - ❑ Eight ancho peppers, dried
  - ❑ Eight guajillo peppers, dried
  - ❑ Two tablespoons of molasses
  - ❑ One tablespoon of cocoa powder
  - ❑ Six oz lager beer
  - ❑ Three tablespoon of cumin
  - ❑ Half cup beef broth
  - ❑ Two cups tomato sauce
  - ❑ One large yellow bell pepper, diced
  - ❑ One large jalapeno pepper, diced
  - ❑ One cup light kidney beans
  - ❑ One cup dark kidney beans
  - ❑ Three Bay leaves
- ◆ **instructions:**
  - ❑ Combine the dried peppers in a food processor and process for two minutes.
  - ❑ Add crushed garlic, molasses, and cocoa powder, and process for two minutes.
  - ❑ Add oil to a large Dutch oven over medium low heat and heat up for three to four minutes.

- ❑ Turn the heat to medium, add onions and cook, stirring frequently, until it just starts to caramelize for about four to eight minutes.
- ❑ Add ground beef to onions, stirring frequently until meat is browned, for about eight minutes.
- ❑ Combine the dried chilies mixture with ground beef and onions, and sauté for three to four minutes.
- ❑ Add beer and stir to loosen any browned bits from the bottom of the Dutch oven, and simmer over medium heat for five minutes.
- ❑ Add tomato sauce and cumin, and stir until combined. Simmer for five minutes.
- ❑ Add diced pepper, kidney beans, and bay leaves. Reduce heat to low and simmer, stirring occasionally, for 30 minutes.

## **Contributing the Views semantic tabs module to Drupal**

In this chapter, we have put a lot of effort in the semantic tabs module. It seems that it would make a lot of sense to make these enhancements available to the Drupal community as a whole. But, before we do that, there are a few things we need to do to ensure that the module is as useful as possible for the Drupal community.

Drupal has coding standards that are strictly enforced when promoting any code for the first time. A good overview of coding standards for Drupal is available at <http://drupal.org/coding-standards>. Before any code is contributed to Drupal, it should be checked to make sure that it conforms to Drupal's coding standards. Thankfully, this is pretty easy, because as pointed out on the page mentioned previously, there is a Coder module that provides an automated process for checking standards compliance of your code.

However, we don't need to install this module as we have online tools available to do this job. <http://pareview.sh> is a service which performs automated reviews of Drupal projects using PHP CodeSniffer. This online service provides the recent pareview script without installing a local testing environment. Now it's time to create a sandbox module project in Drupal and push our views semantic tabs module code.

## Time for action – creating a sandbox for the views semantic tabs module

Drupal allows us to create a new project type such as a module, theme, or distribution. In our case, we want to contribute a module.

1. Go to <https://www.drupal.org/node/add> and click on the **Module project** link.

The screenshot shows the Drupal.org website's 'Add content' page. It features a blue header with the Drupal logo and a search bar. Below the header, there are links for 'View Profile', 'Dashboard', and 'Logout'. The main content area has a title 'Add content' and a list of project types:

- Book listing**: A published published print or e-book covering a Drupal-related topic.
- Book page**: A book page is a page of content, organized into a collection of related entries collectively known as a *book*. A *book page* automatically displays links to adjacent pages, providing a simple navigation system for organizing and reviewing structured content.
- Case study**: Case studies used to feature well-built and well-designed Drupal sites.
- Change record**: Used to record a change in a project, such as an API change.
- Distribution project**: Distributions provide site features and functions for a specific type of site as a single download containing Drupal core, contributed modules, themes, and pre-defined configuration. They make it possible to quickly set up a complex, use-specific site in fewer steps than if installing and configuring elements individually.
- Drupal core**: The Drupal core system or one of the experimental clones of it.
- Issue**: An issue that can be tracked, such as a bug report, feature request, or task.
- Module project** Extend and customize Drupal functionality with contributed modules. If a module doesn't quite do what you want it to do, if you find a bug or have a suggestion, then [join forces](#) and help the module maintainer. Or, share your own by [starting a new module](#).
- Organization**: Listings for the marketplace section of the site.
- Theme Engine project**: Theme engines are how themes interact with Drupal. Use the default theme engine included with Drupal core unless you are using a theme that specifically requires a different theme engine.
- Theme project**: Themes allow you to change the look and feel of your Drupal site. You can use themes contributed by others or create your own to share with the community. Contributed themes are not part of any official release and may not have optimized code/functionality for your purposes.

2. I'm filling out the form. Our new project will initially be a Sandbox project. And I already have permission to promote projects from Sandbox to Full projects. I can see a checkbox allowing you to choose **Full project**, but normally it is best to start with a Sandbox anyway.

The screenshot shows a 'Create Module project' form. At the top, there are links for 'View Profile', 'Dashboard', and 'Logout'. Below that, the title 'Create Module project' is displayed. The form fields include:

- Name \***: Views semantic tabs
- Project type \***:  
A dropdown menu showing 'Sandbox project' with a red arrow pointing to it. A tooltip message states: 'This field visible only if you have permission to promote projects from Sandbox to Full projects.'
- Maintenance status \***:  
A dropdown menu showing 'Actively maintained'.
- Development status \***:  
A dropdown menu showing 'Under active development'.
- Module categories**:  
A dropdown menu listing 'User Access & Authentication', 'User Management', 'Utility', and 'Views'.
- Images**:  
A section for adding files, showing a 'Choose File' button, a file input field ('No file chosen'), and an 'Upload' button. It includes file size restrictions ('Files must be less than 50 MB.') and allowed file types ('Allowed file types: png gif jpg jpeg').
- Description (Edit summary)**:  
A rich text editor with a toolbar containing icons for bold, italic, underline, etc. The text area contains code snippets and notes about configuration and examples.

3. By clicking on the **Save** button, Drupal creates and loads a page for your new project: <https://www.drupal.org/sandbox/krishnakanth17/2665888>.

The screenshot shows a project page titled 'Views semantic tabs' by 'krknth'. The page has tabs for 'Download & Extend Home', 'Drupal Core', 'Distributions', 'Modules' (which is selected), and 'Themes'. Below the tabs, there's a 'View' tab highlighted in green. The page content includes a note about it being an 'Experimental Project' and containing experimental code for developer use only. It describes the plugin as providing a views style plugin to display views results in jQuery UI Tabs. A section titled 'Mandatory notes for configuration:' lists instructions for configuring group fields, specifically mentioning to uncheck 'Link to the Content' checkbox if 'Formatter' is not a link type. Example: If 'title' is selected as a group field, ensure it's not checked for 'Link to the Content'.

4. Click the **Version control** tab near the top of the new project page for instructions on how to start committing code to your sandbox repository.

This screenshot shows the same project page as above, but with the 'Version control' tab highlighted with a red box. The page content is identical to the previous screenshot, including the experimental status and configuration notes. A callout box on the right side highlights the 'Maintainers for Views semantic tabs' section.

5. Setting up this repository for the first time, there are a few steps we need to follow with Git to push our module code. Note: you will be prompted to enter your Drupal password after the last step.

```
$ mkdir views_semantic_tabs  
$ cd views_semantic_tabs  
$ git init
```

```
$ git checkout -b 8.x-1.x
$ echo "name = Views semantic tabs" > views_semantic_tabs.info.yml
$ git add views_semantic_tabs.info
$ git commit -m "Initial commit."
$ git remote add origin krishnakanth17@git.drupal.org:sandbox/
krishnakanth17/2665888.git
$ git push origin 8.x-1.x
```

6. Next, all our module files need to be pushed to the sandbox. So copy and paste all the files inside this directory and follow these git commands as per the <https://www.drupal.org/project/2665888/git-instructions> page.

```
$ git add -A
$ git commit -m "Pushing all files of module"
$ git push -u origin 8.x-1.x
```

7. We are done with creating the sandbox project and pushing our module code to it. Next, we need to promote this module to the full project. I have permission to promote projects from Sandbox to Full projects, once our sandbox is in a state where we think it is ready to be promoted to a full project, can apply for that permission. If we don't have this permission then we must go through a one-time approval process. Check <https://www.drupal.org/node/1011698> for more information.
8. But we are proceeding to promoting to the full project, our code is contributed to drupal.org, and it should be checked to make sure that it conforms to Drupal's coding standards. So we will be using online tools to check Drupal standards. As we discussed earlier, we will be using <http://pareview.sh>. In the URL input box, enter the URL <http://git.drupal.org/sandbox/krishnakanth17/2665888.git> and click on the **Submit branch** button.

## Review a drupal.org project online

Review a branch

URL to git repository [optional: branch name] \*

Example usage:  
<http://git.drupal.org/sandbox/patrickd/1182300.git>  
<http://git.drupal.org/project/fail.git>  
<http://git.drupal.org/project/fail.git 7.x-1.x>

- 9.** After submitting, it takes me to the <http://pareview.sh/pareview/httpgitdrupalorgsandboxkrishnakanth172665888.git> page and it listed the things that need to be fixed.

<http://git.drupal.org/sandbox/krishnakanth17/2665888.git>

[View](#) [Source](#) [Repeat review](#)

Keep in mind that this is primarily a high level check that does not replace but, after all, eases the review process. There is no guarantee that no other issues could show up in a more in-depth manual follow-up review. This service depends on modules that are still under development and changed frequently hence the quality of automated reviews can vary from day to day.

Submitted by Anonymous (not verified) on Fri, 02/19/2016 - 06:52

Git errors:

- Git default branch is not set, see the [documentation on setting a default branch](#).

Review of the 8.x-2.x branch (commit 5d2bd3a):

- [Coder Sniffer](#) has found some issues with your code (please check the [Drupal coding standards](#)). See attachment.
- [ESLint](#) has found some issues with your code (please check the [JavaScript coding standards](#)).

```
js/views-semantic-tabs.js: line 1, col 2, Error - Use the function form of "use strict". (strict)
js/views-semantic-tabs.js: line 2, col 15, Error - Missing space before function parentheses. (space-before-function-paren)
js/views-semantic-tabs.js: line 3, col 10, Error - There should be no spaces inside this paren. (space-in-parens)
js/views-semantic-tabs.js: line 3, col 34, Error - There should be no spaces inside this paren. (space-in-parens)
js/views-semantic-tabs.js: line 4, col 6, Error - Expected indentation of 6 characters but found 4. (indent)
js/views-semantic-tabs.js: line 5, col 2, Error - Newline required at end of file but not found. (eol-last)
js/views-semantic-tabs.js: line 5, col 2, Error - Expected indentation of 2 characters but found 0. (indent)
```

7 problems

- [DrupalPractice](#) has found some issues with your code, but could be false positives.

```
FILE: /var/www/drupal-7-pareview/pareview_temp/views_semantic_tabs.module
-----
FOUND 0 ERRORS AND 1 WARNING AFFECTING 1 LINE
```

- 10.** It will take some time to fix all those errors.

In the next chapter, we will promote this sandbox module to a full project.

## **Summary**

In this chapter, we learned a lot about Views and saw how Views allows you to add interesting components to your site through a web-based user interface. We also learned that Views offers a powerful development platform for custom extensions. We created a sandbox project for the views semantic tabs module. We also looked into some online tools to review our module to check Drupal coding standards.

In the next chapter, we are going to add some visually-striking banner components that will leverage the Views development from this chapter, and show off all the beautiful photos of the recipes on the d8dev site. Finally, we will promote the views semantic tabs module as the full project.



# 10

## Drupal Project Management and Collaboration

*In this chapter, we will learn the process for promoting Sandbox modules to full project status. In doing so, you will learn more about the Drupal developer community, developer collaboration, and how you can become more involved. We will also look into how to manage incremental development on a production site using newly introduced Configuration Management as well as Features. Finally, we will create a rotating banner module for Drupal 8.*

The following topics will be covered in this chapter:

- ◆ Release management in Drupal
- ◆ More advanced module development
- ◆ How to use the Features module

### Rotating banners with the Views Slideshow module

We will examine an approach that is based on a Views plugin and predominantly consisting of Views configuration. Customization of the Views plugin output will be handled with custom CSS.

The Views Slideshow module is an excellent example of a Views style plugin, and it provides much more functionality than just rotating banners. Basically, the Views Slideshow module wraps the jQuery Cycle plugin as a Views style plugin, but it does so with a submodule, which is the `views_slideshow_cycle` module. The `views_slideshow` module is more than a Views style plugin. It's a plugin framework that integrates different jQuery slideshow plugins with Views and also provides a default implementation based on the jQuery cycle plugin.

## Time for action – installing the Views Slideshow module

Before we build a rotating banner with Views, we need to install the Views Slideshow module:

1. Open the Terminal (Mac OS X) or Command Prompt (Windows) application, and change to the root directory of the d8dev site.
2. Use Drush to download and enable the Views Slideshow module:

```
$ drush dl views_slideshow
Project views_slideshow (8.x-4.0) downloaded to /var/www/d8dev/
sites/all/modules/views_slideshow. [success]

Project views_slideshow contains 2 modules: views_slideshow_cycle,
views_slideshow.

root@vb:/var/www/html/drupal8_3_new# drush en views_slideshow
views_slideshow_cycle,
jQuery Cycle appears to be already installed. [ok]
Directory libraries/json2 was created [success]
The latest version of JSON2 has been downloaded to libraries/json
2 [success]
Directory libraries/jquery.hoverIntent was created [success]
The latest version of jQuery HoverIntent has been downloaded to
libraries/jquery.hoverIntent [success]
Directory libraries/jquery.pause was created
[success]
The latest version of jQuery Pause has been downloaded to
libraries/jquery.pause [success]
The following extensions will be enabled: views_slideshow, views_
slideshow_cycle

Do you really want to continue? (y/n): y
views_slideshow was enabled successfully. [ok]
views_slideshow_cycle was enabled successfully.
[ok]
```

## **What just happened?**

Views Slideshow consists of two modules: the base `views_slideshow` module and the `views_slideshow_cycle` module. We need to tell Drush to install the `views_slideshow_cycle` module and Drush will automatically install any dependencies belonging to the same parent module; in this case, the `views_slideshow` module will automatically be enabled by Drush. Also notice how Drush prompted us to download other unmet dependencies for `views_slideshow_cycle`; in this case, the external js libraries `jquery`, `cycle`, `jquery.hoverIntent`, `jquery.pause`, and `json2` are downloaded in the `libraries` folder.

## **Creating a rotating banner with Views Slideshow**

The Views Slideshow module is a Views style plugin. We are going to create a block-based view that will use this style plugin to convert our recipe images list into a rotating banner. After this, we will be able to display it on the front page of our d8dev site.

### **Time for action – creating a banner using the Views Slideshow module**

Now that we have installed and set up the Views Slideshow module, it is time for us to build a Views-based rotating banner:

1. Open the d8dev site in your browser, click on the **Structure** link in the **Admin** toolbar, and then click on the **Views** link.
2. We are creating a new view. Click on the **Add new view** link at the top of the **Views List** page.
3. Enter `Front Banner` as the **View name** and select **Recipe** for the type. We are going to create our rotating banner as a block, so check the **Create a block** checkbox.

4. Next, select **Slideshow** of **fields** for the **BLOCK DISPLAY SETTINGS**. Verify that the Add new view form looks similar to the following screenshot, and click on the **Save and edit** button:

The screenshot shows the 'Add new view' form for creating a 'Front Banner' view. The 'BLOCK DISPLAY SETTINGS' section is highlighted, specifically the 'Display form' dropdown which is set to 'Slideshow' and the 'of:' dropdown which is set to 'fields'. Red arrows point to both of these dropdowns. Other sections visible include 'VIEW BASIC INFORMATION', 'VIEW SETTINGS', 'PAGE SETTINGS', and 'BLOCK SETTINGS'.

**VIEW BASIC INFORMATION**

**View name \***  
Front Banner Machine name: front\_banner [Edit]  
 Description

**VIEW SETTINGS**  
Show: Content of type: Recipe sorted by: Newest first

**PAGE SETTINGS**  
 Create a page

**BLOCK SETTINGS**  
 Create a block  
Block title: Front Banner

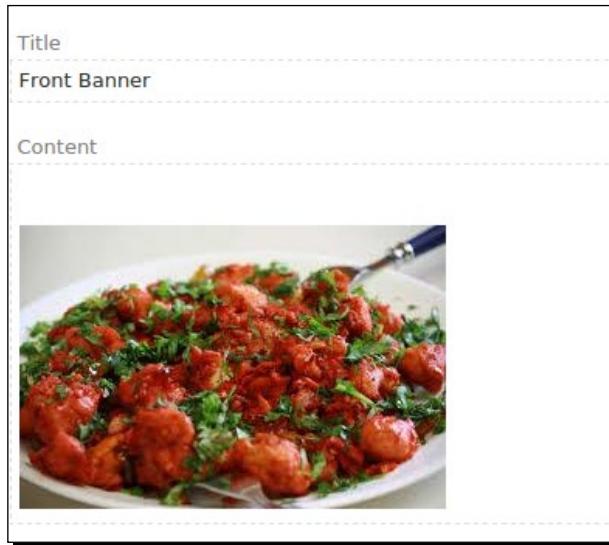
**BLOCK DISPLAY SETTINGS**  
Display form: Slideshow of: fields

Items per block: 5  
 Use a pager

**Save and edit** **Cancel**

5. Now, we need to decide what fields we want to display in the banner. The **Content: Title** field has been added by default. But we obviously want to display an image in the rotating banner.
6. Click on the **Add** button for **FIELDS**, search for **Images** in the **Search** input, and select **Images**. Click on the **Apply (all displays)** button.
7. Next, in the **Configure field** form, select **Large** as the **Image style** and click on the **Apply (all displays)** button.

8. Now, if you scroll down to the **Preview** area of the **Views** page, you will see a working slide show that looks something similar to the following screenshot:



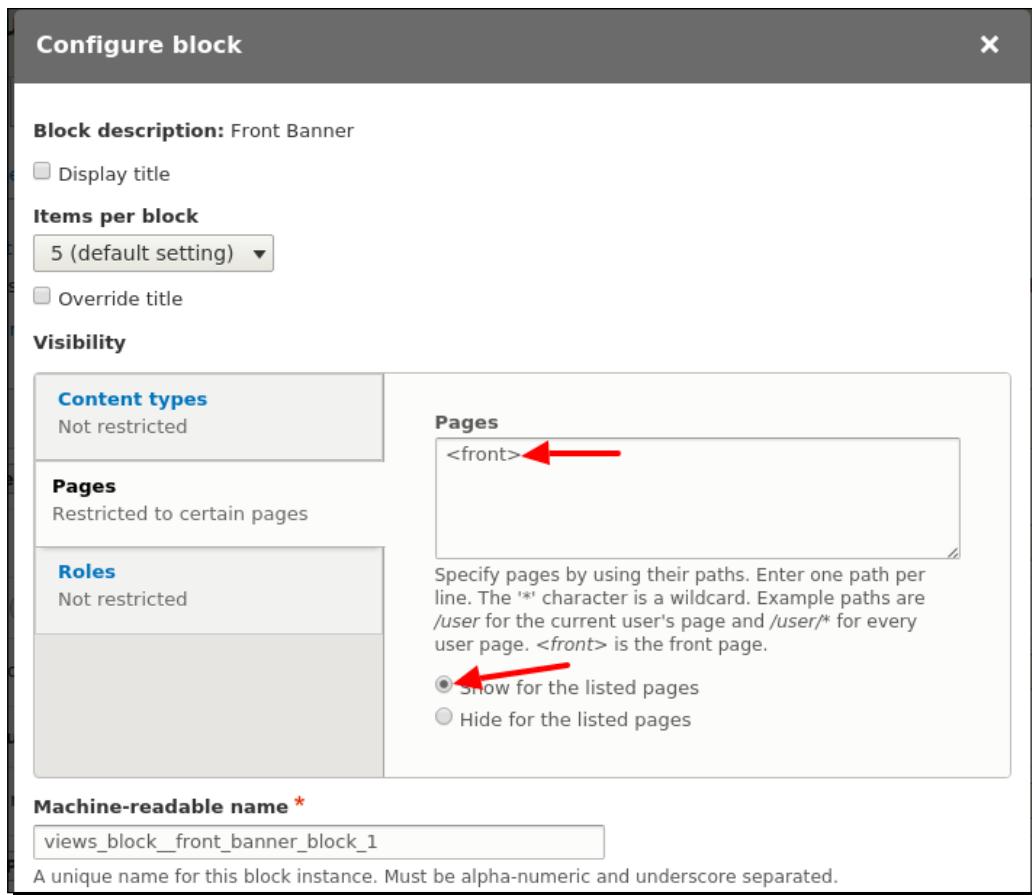
9. Click on the **Add** button for **FILTER CRITERIA**. Search for images in the search input textbox and select the **Any Images (field\_images:title)** filter. Click on the **Apply (all displays)** button.
10. On the next screen, select the **Is not empty (NOT NULL)** operator, and click on the **Apply (all displays)** button.
11. Click on the dropdown for the **FIELDS Add** button and select **Rearrange**:



- 12.** Next, just drag the **Content: Title** field under the **Content: Images** field, and click on the **Apply (all displays)** button.

Now, we are ready to see how our new Views Slideshow banner looks on the front page.

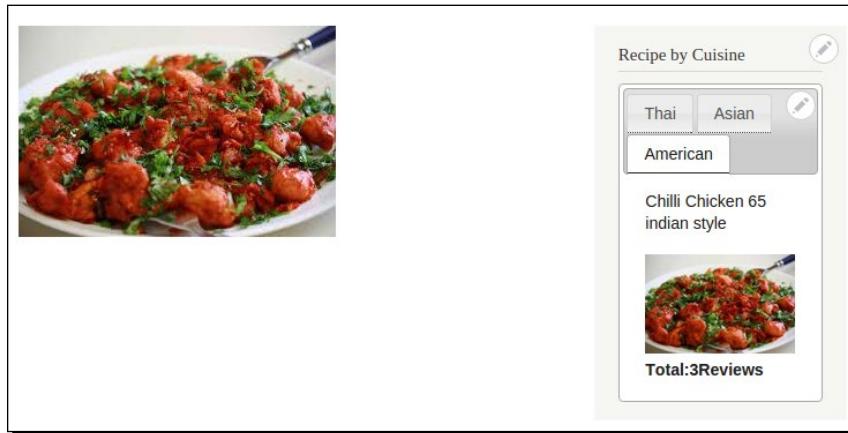
- 13.** Click on the **Save** button for our new view. Then click on the **Structure** link in the **Admin** toolbar, and finally click on the **Block layout** link.
- 14.** Scroll down towards the **Content Region** and click on the **Place block** link. In the popup, search for **Front banner** and click on the **Place block** button for the **Front Banner** block.
- 15.** On the next screen, uncheck the **Override title** checkbox. Next, select **Content** as the region to display the block for our D8Dev. In the **Visibility | pages** section, select **Show for the listed pages**, and enter `<front>` as the only page to display it on:



- 16.** Click on the **Save block** button. On the **Blocks** configuration page, drag the **Front Banner** block above the **Main page content** block in the **Content regions**, and click on the **Save blocks** button at the bottom of the screen.

| Block                 | Type          | Region  | Configure |
|-----------------------|---------------|---------|-----------|
| Front Banner*         | Lists (Views) | Content | Configure |
| Help                  | Help          | Content | Configure |
| Page title            | core          | Content | Configure |
| Primary admin actions | core          | Content | Configure |
| Tabs                  | core          | Content | Configure |
| tab                   | Lists (Views) | Content | Configure |
| Main page content     | System        | Content | Configure |

- 17.** When you are done, the Front Banner should look similar to the following screenshot:



## What just happened?

We created a Front banner block with the Views Slideshow style plugin and assigned the block to the front page of our d8dev site.

We have created a slideshow, but it is displaying the images of size that we uploaded with the recipe. This might be of varying sizes. To make the image size uniform, we will have to create an image style that can be applied to this slide show as well as any other location where we want to display images of the same size.

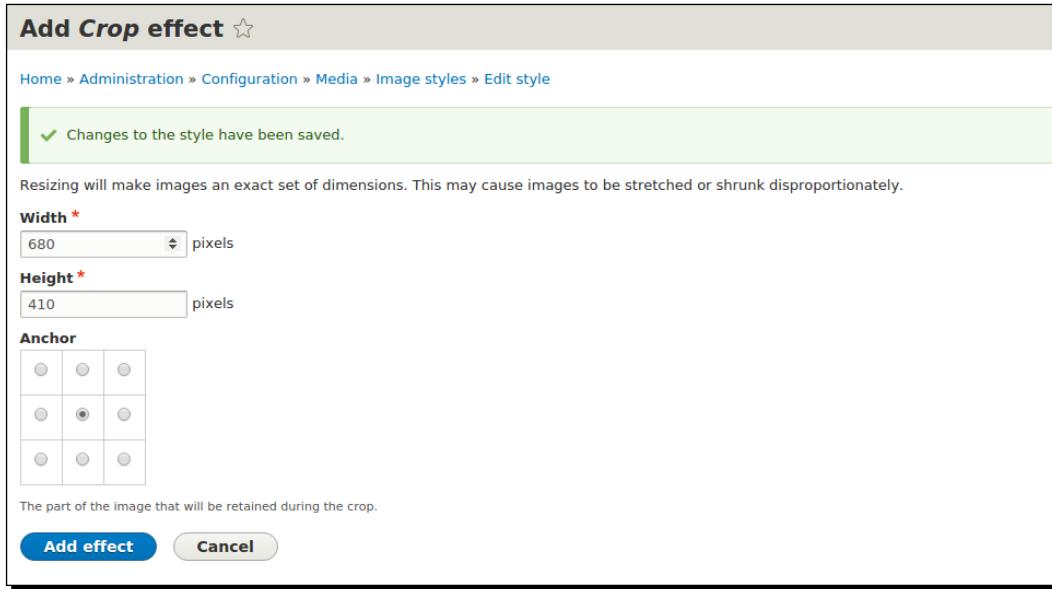
## Time for action – creating a new image style for images in our rotating recipe banner

In the last chapter, we learned how to add image styles for Drupal 8. Let's add a new image style named `front_banner` that will scale our recipe images to be no wider than 680 pixels and cropped to 410 pixels in height. We will apply the same to the image field of our Front Banner view. This will create a more consistent look for our rotating banner as it will not change the size from slide to slide:

1. Open the d8dev site in your browser, click on the **Configuration** link in the **Admin** toolbar, and then click on **Image styles** under the **Media** link.
2. We are creating a new Image style. So, click on the **+Add Image styles** link at the top of the **Image styles** page.
3. Enter **Front banner** as the image style name and click on the **Create new style** button.
4. Select **Crop** as the effect and click on the **Add** button at the bottom of the page:

The screenshot shows the 'Image styles' configuration page. At the top, there is a success message: 'Style Front Banner was created.' Below this, there are two image previews: 'original (view actual size)' and 'Front Banner (view actual size)'. The 'Front Banner' preview is scaled down to 600px width and 410px height. The 'Image style name' field is set to 'Front Banner'. The 'Effect' dropdown is set to 'Crop'. A red arrow points to the 'Crop' button. At the bottom, there are 'Update style' and 'Delete' buttons.

5. Next, Enter 680 as **Width** and 410 as **Height**, and click on the **Add effect** button:



6. Now we have to update this image style for the Front banner view. Click on the **Configuration** link in the **Admin** toolbar, and then click on the **Views** link.
7. Next, click on the **Edit** link for the Front banner view. In the next view editing page, click on the **Content: Images** link for **Fields** and select **Front banner** as the **Image style** field. Click on the **Apply (all displays)** button. Then save the view by clicking on the **Save** button at the bottom.
8. When you are done, the Front Banner should look similar to the following screenshot:



## **What just happened?**

We just created a new image style for images in our rotating recipe banner.

## **Enhancing the appearance of our front banner with a pager and CSS**

Our new front banner works fine, but we can easily improve its appearance and user experience by adding a few more configurations and CSS. We are going to add custom CSS to our d8dev module to tweak the appearance of the rotating banner, but first we are going to add a pager that will show how many slides there are and the current slide.

### **Time for action – updating the front banner view to include a slide show pager**

We are going to enhance our Views rotating banner with a pager:

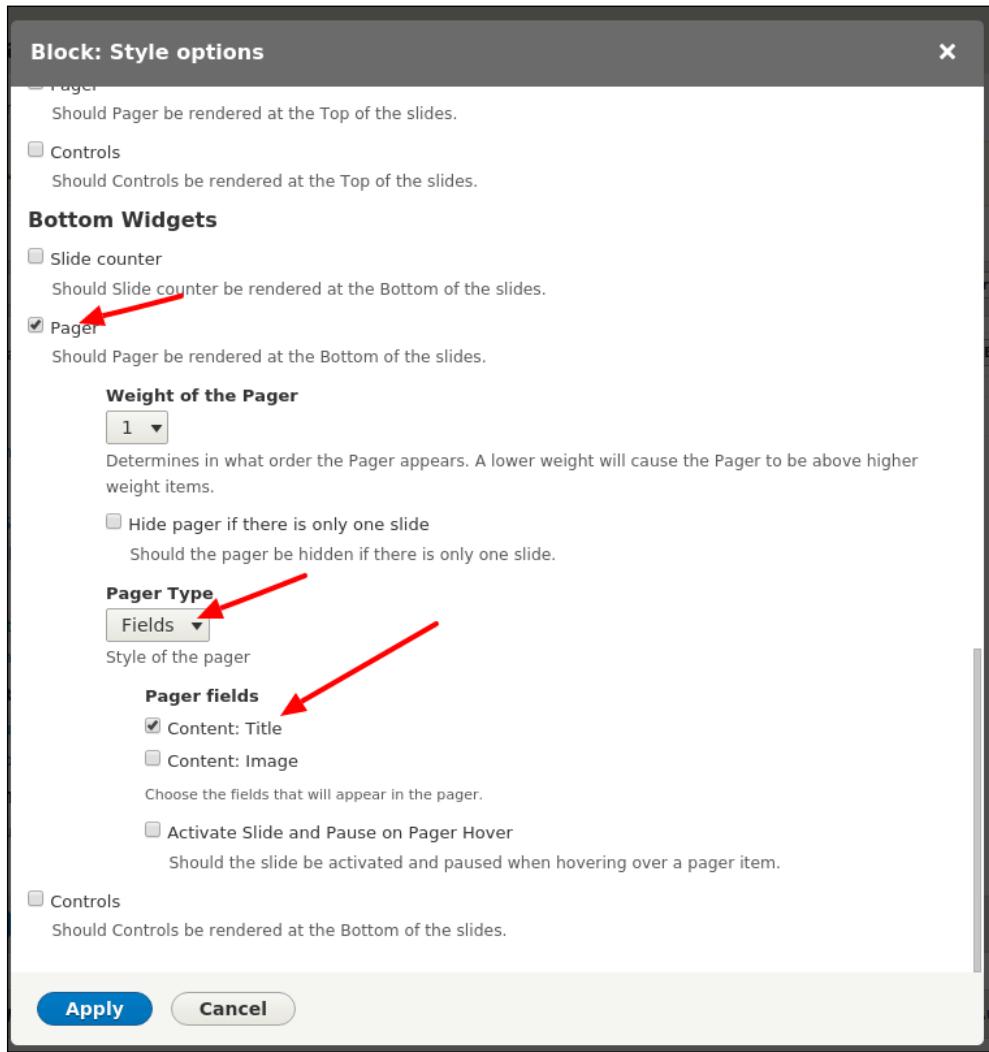
1. Open the d8dev site in your browser, hover the mouse over the new front banner, click on the contextual links widget, and then click on the **Edit view** link:



2. Next, we need to add a pager to our view. Click on the **Settings** link for the slideshow formatter:



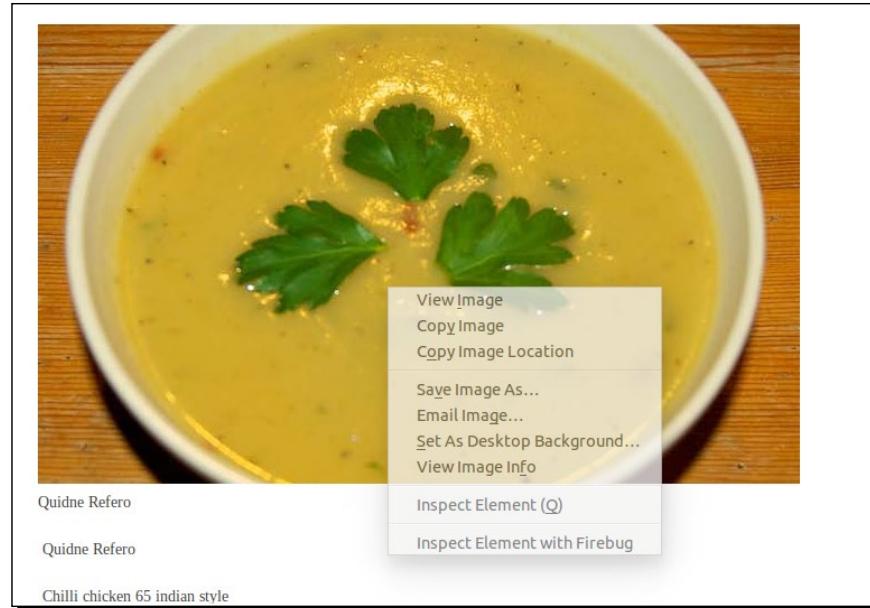
3. In the **Block: Style options** form, scroll down to the **Bottom Widgets** section and check the **Pager** checkbox. After checking this, we can see other fields appear. Check **Content: Title** under the **Pager fields** section and click on the **Apply** button. See this screenshot:



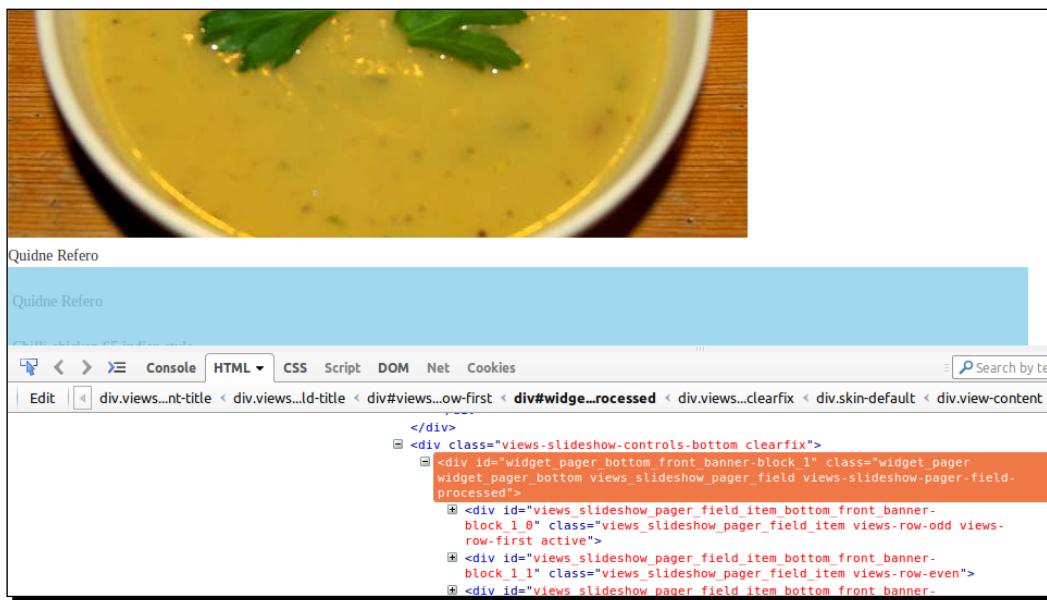
4. Now, click on the **Save** button for the view, and take a look at the updated front banner. See the following screenshot:



5. Though it is not quite the visually striking pager we were looking for, if you click on any of the titles, you will notice that the slide will change to that paged item. So, although the pager works, it does not look all that great. Let's see what we can do about the way it looks by adding some custom CSS to our d8dev module.
6. Open the front page of the d8dev site in Chrome/Firefox, right-click on your rotating banner, and select **Inspect Element** from the contextual menu that pops up:



7. In the **Elements** inspector, find the `div` tag with the `views-slideshow-controls-bottom` class, and expand it:



- 8.** Select the `div` tag with the `views-slideshow-pager-field` class so that it is highlighted. Then click inside the `element.style` curly brackets in the Styles inspector, and type in `float: right;`. We can see changes in the page field titles that floated to the right.
- 9.** Next, in the PhpStorm IDE, open the `d8dev.css` file located at `d8dev/modules/d8dev/styles/`.
- 10.** Scroll to the bottom of the file, and add the following style:

```
div.views_slideshow_pager_field {  
    bottom: 30px;  
    float: left;  
    position: relative;  
    text-align: right;  
    z-index: 113;  
}
```

- 11.** Now, back in the browser, expand the `views-slideshow-pager-field` `div`, find the `div` tag with the `div.views_slideshow_pager_field_item` selector, and add the following styles to the `d8dev.css` file for the `div.views_slideshow_pager_field_item` selector:

```
div.views_slideshow_pager_field_item{  
    display: inline-block;  
    background-color: #999;  
    width: 10px;  
    height: 10px;  
    text-indent: -9999px;  
    border: 2px solid #CCC;  
    -moz-border-radius: 4px;  
    border-radius: 8px;  
    margin-left: 4px;  
    cursor: pointer;  
}  
  
div.views_slideshow_pager_field_item:hover,  
div.views_slideshow_pager_field_item.active{  
    background-color: #BF0000;  
}
```

- 12.** Clear the cache and refresh the front page. See this screenshot:



- 13.** Now add the CSS to hide the views-content-title div. Open the PhpStorm IDE and add the following CSS to style.css, at the end of the file:

```
div.views-content-title {  
    display: none;  
}
```



Note: by excluding the content title from the views field settings will also hide the title, but make sure other CSS will be affected by this.



- 14.** Clear the cache and refresh the front page:



**15.** Now, we need to do some styling for the recipe title. We are going to increase the font size and position it above the pager. But text align in the center of the image and add a background color tomato, with text color white. Also we are updating width as 672 px since we are taking padding as 4 px.

**16.** Add the following CSS to the `div.views-field-title` selector in the PhpStorm IDE:

```
#views_slideshow_cycle_main_front_banner-block_1 .views-field-title {  
    background: tomato none repeat scroll 0 0;  
    bottom: 40px;  
    color: white;  
    display: block;  
    font-weight: bold;  
    padding: 4px;  
    position: absolute;  
    text-align: center;  
    width: 672px;  
}
```

**17.** Clear the cache and refresh the front page in the browser; you will see that the recipe titles are much easier to read.



Chilli chicken 65 Indian style

## ***What just happened?***

We added a pager and caption to our Front banner, and although we did not write much custom PHP code, we saw how a little bit of Views configuration with the right contrib module, Views Slideshow, and some creative CSS can be combined to great effect.

## **Time for another recipe**

Here is a hearty and tasty soup for a cold winter's day. Just about anyone, with just about any dietary restrictions, should be able to enjoy this healthy and delicious soup.



- ◆ **Name:** Potato Leek Soup (Vegan)
- ◆ **cuisineType:** European
- ◆ **description:** This healthy yet still creamy soup will really stick to your ribs and warm you up on a cold day.
- ◆ **recipeYield:** Ten servings
- ◆ **prepTime:** 30 minutes
- ◆ **cookTime:** 45 minutes

◆ **ingredients:**

- Five to six large russet potatoes, peeled and quartered
- Four leeks, cleaned and thinly sliced
- One large sweet onion, diced
- Four tablespoon vegan butter
- One tablespoon olive oil
- Six cups vegetable broth
- One cup plain soy milk
- Sea salt
- Freshly ground black pepper
- One tablespoon rice vinegar
- Two teaspoon crushed red pepper
- One-fourth cup parsley, finely chopped

◆ **instructions:**

- Melt the vegan butter in a large Dutch oven over medium heat.
- Once the butter melts, add diced onion and sauté until it just starts to caramelize.
- Add the finely sliced leeks and sauté over medium heat for 10 minutes, stirring every minute or so.
- Add diced potatoes and sauté with leeks and onions for 10 minutes.
- Add the olive oil and stir to combine.
- Add vegetable broth and plain soy milk. Stir to combine. Bring to a boil over medium-high heat, and reduce the heat to low.
- Simmer over low heat for 15 minutes.
- Using an emersion hand blender, blend the soup into a smooth puree.
- Stir in vinegar and crushed red pepper.
- Stir in sea salt and freshly ground black pepper to taste.
- Stir in fresh parsley and enjoy.

## Promoting a sandbox project to a full project

Although we committed our Views semantic tabs module changes to the sandbox Git repository—and in doing so made the code available to anyone who wants to use it—using Git is a barrier for many people who are not developers and just want to download a module, configure it, and use it. A sandboxed module will also deter people from trying your module, because they may not trust a module that is not a full project (and Drupal includes a big warning at the top of all sandboxed module pages). I will create a release that can be downloaded easily without Git.

At this point, I will create only an alpha release until the community has had an opportunity to test it. Once there has been some feedback, I will create a full release. Before we begin, we will follow the Module Documentation Guidelines at <https://www.drupal.org/node/161085>. After reading this page, I have decided to add the `README.txt` file to our views semantic tabs module.

### Time for action – creating `README.txt` and pushing to the sandbox

1. Create the `README.txt` file under the `views_semantic_tabs` folder as follows :

Introduction  
-----

This module provides a views style plugin to display views results in jQuery UI Tabs.

Requirements  
-----

This module requires the following modules:

\* Views (<https://drupal.org/project/views>)

INSTALLATION  
-----

\* Install as you would normally install a contributed Drupal module. See:

```
https://drupal.org/documentation/install/modules-themes/
modules-8
for further information.
```

CONFIGURATION

---

```
* Update your view with semantic tabs style under format style
in the view.
```

MAINTAINERS

---

Current maintainers:

```
* Krishna kanth (krknth) - https://drupal.org/u/krknth
* Neeraj kumar (neerajskydiver) -
https://drupal.org/u/neerajskydiver
```

2. Next, we need to push it to the Git sandbox. Open the terminal, go to the `modules/views_semantic_tabs` folder, and run the following commands:

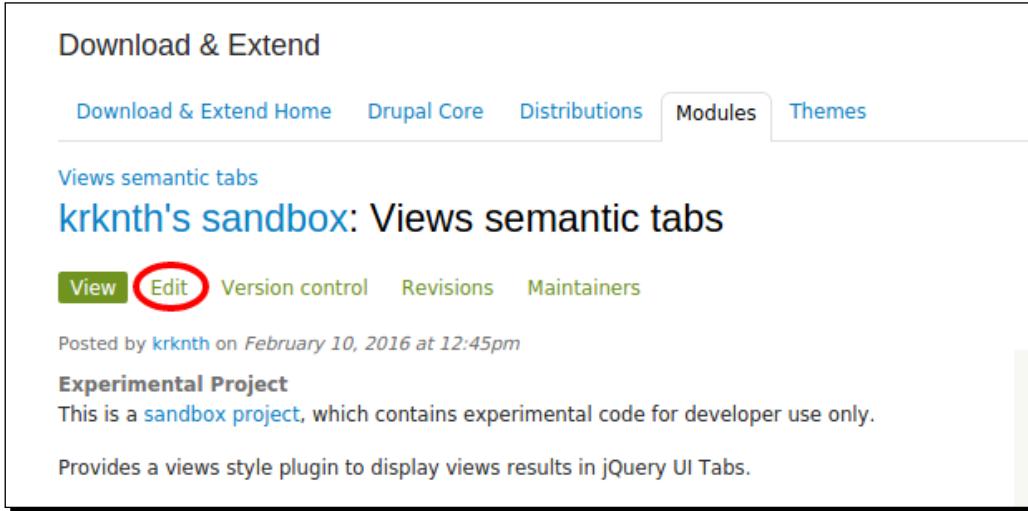
```
$ git add README.txt
$ git commit -m "Added README.txt file." $ git push origin 8.x.2.x
```

## ***What just happened?***

We learned a bit about the Drupal `README.txt` file in the Module Documentation Guidelines. We added it to the module and pushed it to the Drupal project Git repository.

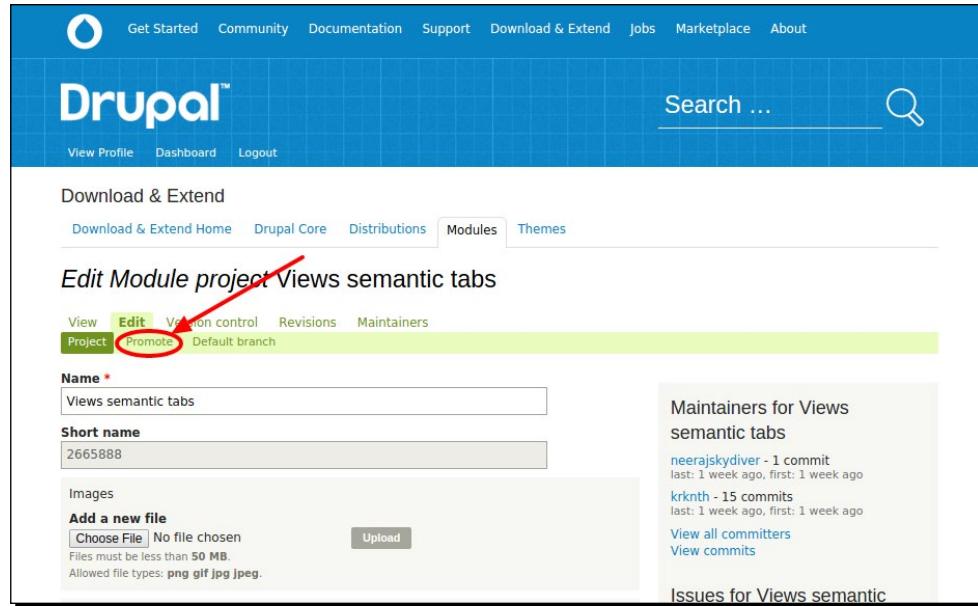
## Time for action – promoting the Views semantic module to a full project on Drupal.org

1. Visit our sandbox project's page at <https://www.drupal.org/sandbox/krishnakanth17/2665888>. And click on the **Edit** tab:



The screenshot shows the 'krknth's sandbox: Views semantic tabs' project page. At the top, there's a navigation bar with 'Download & Extend Home', 'Drupal Core', 'Distributions', 'Modules' (which is selected), and 'Themes'. Below the navigation, it says 'Views semantic tabs' and 'krknth's sandbox: Views semantic tabs'. There are tabs for 'View', 'Edit' (which is circled in red), 'Version control', 'Revisions', and 'Maintainers'. A note below says 'Posted by krknth on February 10, 2016 at 12:45pm'. Under 'Experimental Project', it says 'This is a [sandbox project](#), which contains experimental code for developer use only.' A description below states 'Provides a views style plugin to display views results in jQuery UI Tabs.'

2. Then the **Promote** subtab:

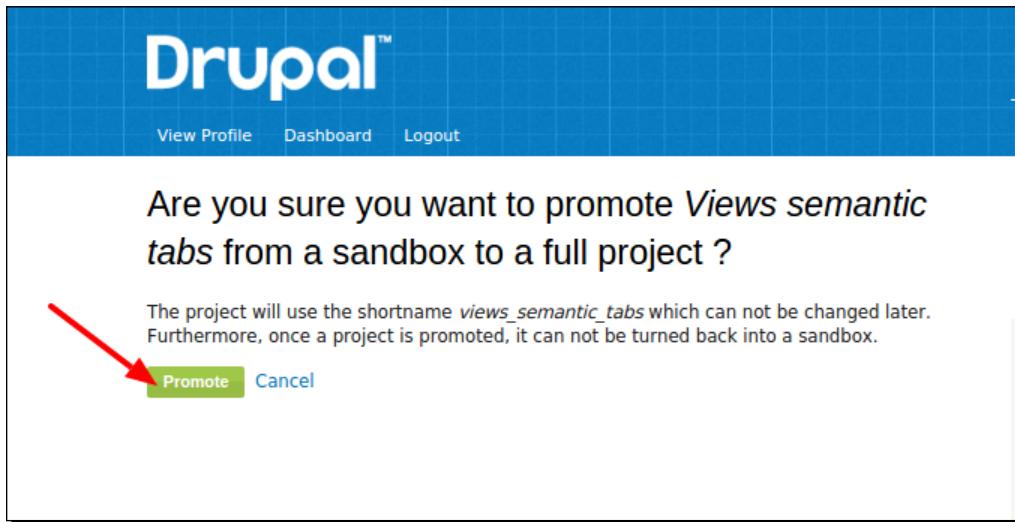


The screenshot shows the 'Edit Module project Views semantic tabs' page. The navigation bar includes 'Get Started', 'Community', 'Documentation', 'Support', 'Download & Extend', 'Jobs', 'Marketplace', and 'About'. Below the navigation, it says 'Download & Extend' and lists 'Download & Extend Home', 'Drupal Core', 'Distributions', 'Modules' (selected), and 'Themes'. The main content area has tabs for 'View', 'Edit' (selected), 'Version control', 'Revisions', and 'Maintainers'. A 'Project' tab is also present, with 'Promote' (highlighted with a red arrow) and 'Default branch'. Form fields include 'Name \*' (Views semantic tabs), 'Short name' (2665888), and an 'Images' section with a file upload field ('Add a new file', 'Choose File', 'No file chosen', 'Upload'). To the right, there's a sidebar for 'Maintainers for Views semantic tabs' showing 'neerajskydiver - 1 commit' and 'krknth - 15 commits'. It also has links for 'View all committers' and 'View commits'. Another sidebar on the right is titled 'Issues for Views semantic'.

3. Fill out the form, making sure to enter a short project name:

The screenshot shows a Drupal project management interface. At the top, there are tabs: View, Edit, Version control, Revisions, Maintainers, Project, Promote, and Default branch. The 'Promote' tab is highlighted. Below the tabs, a confirmation message reads: "I understand that this action cannot be undone and wish to proceed anyway. Please confirm that you understand the implications of promoting this project." A red arrow points to the input field for the "Short project name \*". The value "views\_semantic\_tabs" is entered. A note below says, "You may not edit this value after the project has been promoted." Another red arrow points to the "Enable releases" checkbox, which is checked. A note below it says, "Allow releases of this project with specific versions. This can be changed later." At the bottom is a green "Promote to full project" button.

4. The next page asks whether I am sure that I want to promote the module, and I am sure, so I will click on the **Promote** button:



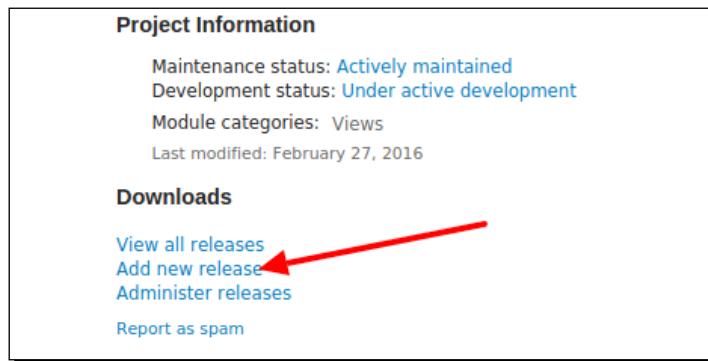
- 5.** Now it is no longer a sandbox project, and Drupal provides some important instructions regarding the remote repository for the project. The Git command in the following screenshot needs to be executed before any new changes can be pushed to the remote repository, because Drupal has moved it to a new location for full projects and to match the new short project name:

The screenshot shows a screenshot of the Drupal 'Download & Extend' interface. At the top, there's a navigation bar with tabs: 'Download & Extend Home', 'Drupal Core', 'Distributions', 'Modules' (which is selected and highlighted in blue), and 'Themes'. Below the navigation, the title 'Views semantic tabs' is displayed. Underneath the title, there are several tabs: 'View' (selected and highlighted in green), 'Edit', 'Version control', 'Revisions', 'Maintainers', and 'Automated Testing'. A green box contains a success message: 'The project *Views semantic tabs* has been promoted to a full project.' and 'New URL'. It also includes instructions: 'Now that this experimental project has been promoted, you'll need to update the URL of your remote repository or reclone it.' and 'Don't forget to substitute your project's new short name in the brackets below (and remove the brackets!)'. Below this box, there's a code snippet: `git remote set-url origin krishnakanth17@git.drupal.org:[project_short_name_here].git`. At the bottom of the screenshot, there's a note: 'Posted by krknth on February 10, 2016 at 12:45pm' and 'Provides a views style plugin to display views results in jQuery UI Tabs.' followed by a bolded 'Mandatory notes for configuration :

- 6.** After all these steps, our module is promoted to a full project and we can access it with the URL [https://www.drupal.org/project/views\\_semantic\\_tabs](https://www.drupal.org/project/views_semantic_tabs).
- 7.** Now we need to create a dev release for the Views semantic tabs module so that it is easier to download and install. Drupal provides instructions on the version control page of a full project page for creating a branch for a dev release with Git (these instructions are only displayed for module maintainers).
- 8.** Executing the following Git commands on our local Views semantic tabs repository will create a new dev branch:

```
$ git checkout -b 8.x-2.x
$ git push -u origin 8.x-2.x
```

- 9.** Now that we have a new dev branch in the Views semantic tabs repository, we will be able to add a dev release to the project. On the **View** page of the Views semantic tabs module, there is an **Add new release** link below **Project information**. Clicking on that link will take us to the **Create Project release** page. See this screenshot:



- 10.** The **Create release** page lists two Git release branches to select from, the **8.x-1.x** and **8.x-2.x** branch we just created. So we just need to select **8.x-2.x (8.x-2.x-dev)** and click on the **Next** button:

A screenshot of a web page titled "Create Release". It features a dropdown menu labeled "Git release tag or branch \*". The dropdown shows "8.x-2.x (8.x-2.x-dev)" as the selected option. Below the dropdown is a small explanatory text: "Select the Git tag or branch (and therefore version number) for this release." At the bottom of the form is a green "Next" button.

- 11.** On the next page, we will enter some **Release notes** and click on the **Save** button; the other fields will already be filled because this is the first dev release.

**Create Release**

**Release identification**  
Now that the branch has been selected, these can not be modified unless you [go back to the previous page](#).

**Git branch \***

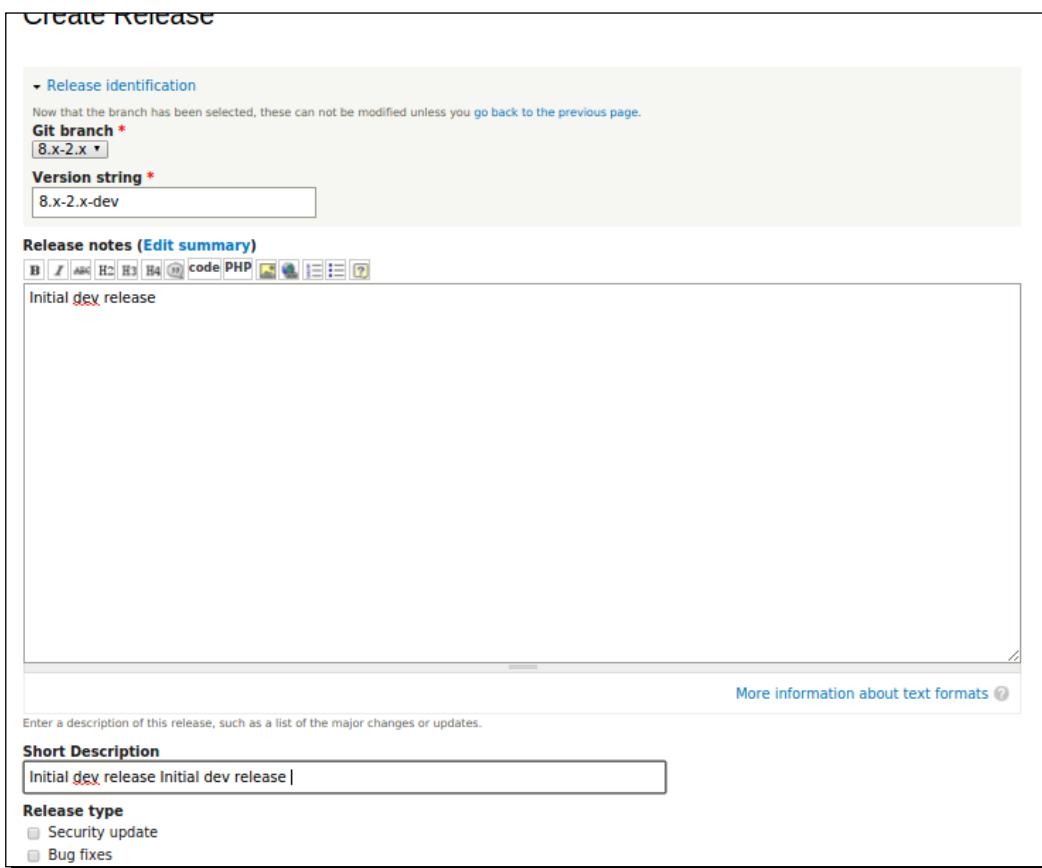
**Version string \***

**Release notes (Edit summary)**  
Enter a description of this release, such as a list of the major changes or updates.  
More information about text formats [?](#)

Initial **dev** release

**Short Description**

**Release type**  
 Security update  
 Bug fixes



- 12.** We have successfully promoted the Views semantic tabs module to full project status, and have created an initial dev release.

The screenshot shows the 'Download & Extend' section of the Drupal interface. In the top navigation bar, 'View Profile', 'Dashboard', and 'Logout' are visible. Below that, 'Download & Extend Home', 'Drupal Core', 'Distributions', 'Modules' (which is selected), and 'Themes' are listed. Under 'Modules', the path 'Views semantic tabs » Releases' leads to the page for 'views\_semantic\_tabs 8.x-2.x-dev'. On this page, there are 'View' and 'Edit' buttons. A green success message box contains the text: 'Release views\_semantic\_tabs 8.x-2.x-dev has been created.' To the left, under 'Release notes', it says 'Initial dev release'. To the right, under 'Release info', it shows: 'Created by: krknth', 'Created on: February 27, 2016 - 15:28', 'Last updated: February 27, 2016 - 15:28', and 'Core compatibility: 8.x'. A link 'View usage statistics for this release' is also present.

- 13.** Drupal will automatically generate the `.tar.gz` and `.zip` files and attach them to the project, but it may take as long as 1-2 hours for a dev release (official non-dev releases are published within 5 minutes). Until then, only an unpublished release node will appear on the Releases for Views semantic tabs page. Also we can use Drush to download and enable the module by using this command:

```
$ drush dl views_semantic_tabs
```

### **What just happened?**

We learned a bit about the Drupal `README.txt` file from the Module Documentation Guidelines. We also learned how a module can be promoted from a sandbox project to a full project and how to release a dev release to the project.

## **Introducing the Features module**

The Drupal 8 Features module enables us to capture and manage a feature, which is a set of Drupal entities. The Features module will take different site-building components from modules by providing a UI and an API with exportables and bundle them together in a single module. A feature in normal terms might be a blog, a page, or an image gallery. So in the next topic, we are going to learn how to use the Features module. We are going to export our Recipe content type and use it in another environment.

## Time for action – installing the Features module

We are going download and install the Features module using Drush. Also we will install features\_ui module as its package:

1. Open the Terminal (Mac OS X) or Command Prompt (Windows) application, and change to the root directory of our d8dev site.
2. Use Drush to download and enable the Features module:

```
$ drush dl features
Project features (8.x-3.0-alpha6) downloaded
to /var/www/html/d8dev/modules/features.
[success]

Project features contains 2 modules: features_ui, features.

$ drush en features
The following projects have unmet dependencies:
[ok]
features requires config_update
Would you like to download them? (y/n): y
Project config_update (8.x-1.0) downloaded to /var/www/html/
drupal8_3_new//modules/config_update.
[success]

Project config_update contains 2 modules: config_update_ui,
config_update.

The following extensions will be enabled: features, config_update
Do you really want to continue? (y/n): y
config_update was enabled successfully.

[ok]

features was enabled successfully.

[ok]

$ drush en features_ui
The following extensions will be enabled: features_ui
Do you really want to continue? (y/n): y
features_ui was enabled successfully.
```

### ***What just happened?***

We enabled the Features module. It consists of two modules: the base Features module and the features\_ui module. The Features module depends on the config\_update module, which is downloaded automatically after enabling the Features module. We also enabled the features\_ui module, which gives an easy UI to create features.

## Recipe feature by the Features module

We are going to learn how to use the Features module by exporting our Recipe content type and related configurations such as—Fields of Recipe content type, Form displays, Form view displays, Views – Front Banner, Random Top Rated Recipe, Top rated recipes And Blocks – Top Recipe, Front Banner. Then we will look into how we can use it in other environments.

### Time for action – using the Features module to export the Recipe content type and related configurations

We are going to export our Recipe content type as a feature module and also export its related configurations as part of the Feature exported module:

1. Open the d8dev site in your browser, click on the **Configuration** link in the **Admin** toolbar, and click on the **Features** link under the **DEVELOPMENT** section. We can see the next page as shown in following screenshot:

The screenshot shows the 'Features' configuration page. At the top, there is a button labeled '+ Create new feature' with a red arrow pointing to it. Below this, there is a 'Bundle' dropdown set to 'Default'. A table lists various features with columns for 'FEATURE', 'DESCRIPTION', 'VERSION', 'STATUS', and 'STATE'. The 'Recipe' feature is highlighted with a red arrow and is listed as 'recipe\_image'. At the bottom of the page, there is a note: 'Use an export method button below to generate the selected features.' followed by two buttons: 'Download Archive' and 'Write'.

| FEATURE          | DESCRIPTION  | VERSION  | STATUS | STATE        |
|------------------|--------------|--|--------|--------------|
| Default comments | comment      | Provides Default comments comment type and related configuration. Allows commenting on content<br>► Included configuration   |        | Not exported |
| Article          | article      | Provides Article content type and related configuration. Use <i>articles</i> for time-sensitive content like news, press releases or blog posts.<br>► Included configuration |        | Not exported |
| Basic page       | page         | Provides Basic page content type and related configuration. Use <i>basic pages</i> for your static content, such as an 'About us' page.<br>► Included configuration          |        | Not exported |
| Recipe-old       | recipe       | Provides Recipe-old content type and related configuration.<br>► Included configuration  |        | Not exported |
| Recipe           | recipe_image | Provides Recipe content type and related configuration.<br>► Included configuration  |        | Not exported |
| User             | user         | Provide User related configuration.<br>► Included configuration  |        | Not exported |
| Core             | core         | Provides core components required by other features.<br>► Included configuration   |        | Not exported |
| Site             | site         | Provides site components.<br>► Included configuration  |        | Not exported |
| Unpackaged       | unpackaged   | Configuration that has not been added to any package.<br>► Included configuration  |        | Not exported |

- 2.** Next click on the **Included configuration** link under the **DESCRIPTION** column for the Recipe feature.

| Recipe                     | recipe_image  | Provides Recipe content type and related configuration.<br>▼ included configuration | Not exported |
|----------------------------|---|---|--------------|
| <b>Block</b>               | tab tab   | Front Banner  |              |
| <b>Content type</b>        | Recipe  |   |              |
| <b>Dependencies</b>        | <a href="#">Block</a> <a href="#">Comment</a> <a href="#">Field</a><br><a href="#">Image</a> <a href="#">Menu UI</a> <a href="#">Node</a><br><a href="#">Path</a> <a href="#">System</a><br><a href="#">Taxonomy</a> <a href="#">Text</a> <a href="#">User</a><br><a href="#">Views</a><br><a href="#">Views semantic tabs</a><br><a href="#">Views Slideshow</a> |   |              |
| <b>Entity form display</b> | node.recipe_image.default   |   |              |
| <b>Entity view display</b> | node.recipe_image.default<br>node.recipe_image.teaser   |   |              |
| <b>Field</b>               | recipeCuisine Image<br>Comment Body   |   |              |
| <b>Field storage</b>       | node.field_recipecuisine  |   |              |
| <b>Taxonomy vocabulary</b> | Type of Cuisine   |   |              |
| <b>View</b>                | tab   | Front Banner  |              |

- 3.** Here we can understand what configurations are included for this feature. It has the two blocks (Top Recipe and Front Banner), three views, and other field-related configurations that we are looking for. But I have noticed that the **Recipes by Cuisine** block is missing, so we will add this missing component in the next step.
- 4.** Now click on the **Recipe** link feature. In the next page, we can see a list components and general information about this feature:

**GENERAL INFORMATION**

**Name**: Recipe  
Machine name: recipe\_image [Edit]  
Example: Image gallery (Do not begin name with numbers.)

**Description**: Provides Recipe content type and related configuration.

Provide a short description of what users should expect when they install your feature.

**Bundle**: Default

**Version**: Examples: 8.x-1.0, 8.x-1.0-beta1  
Mark all config as required  
Required config will be assigned to this feature regardless of other assignment plugins.

Allow conflicts  
Allow configuration to be exported to more than one feature.

**Download Archive** **Write**

**COMPONENTS**

Search  Clear  Select all

- ▶ Simple configuration (31)
- ▶ Action (14)
- ▶ Base field override (2)
- ▶ Block (23)
  - Front Banner (views\_block\_front\_banner\_block\_1)
  - tab (views\_block\_tab\_block\_1)
  - tab (views\_block\_tab\_block\_1\_2)
- ▶ Comment type (1)
- ▶ Contact form (2)
- ▶ Content type (3)
  - Recipe (recipe\_image)
- ▶ Custom block type (1)
- ▶ Date format (11)
- ▶ Entity form display (6)
  - node.recipe\_image.default

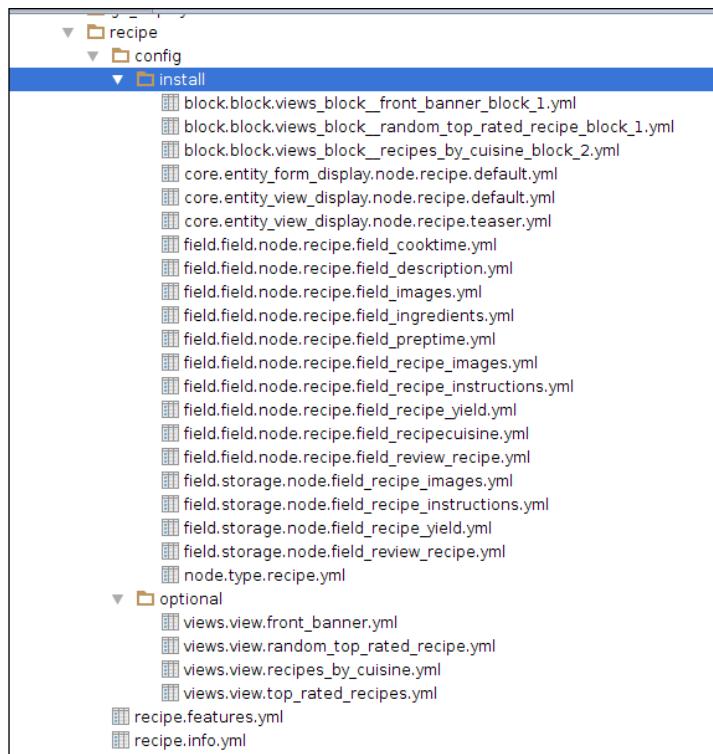
5. Next under the **COMPONENTS** section, search for `recipes`. We notice that **Recipes by Cuisine: Block 2 (views\_block\_recipes\_by\_cuisine\_block\_2)** is visible under the **Block** section and select that checkbox:

The screenshot shows the 'COMPONENTS' search results. A red arrow points to the search bar containing 'recipe'. Another red arrow points to the 'Block' section, which contains several items. One item, 'Front Banner (views\_block\_front\_banner\_block\_1)', has a checkbox next to it that is checked. Other items in the 'Block' section include 'tab (views\_block\_tab\_block\_1)' and 'tab (views\_block\_tab\_block\_1\_2)', both of which also have checked checkboxes.

6. Also, select the **Recipes by Cuisine (recipes\_by\_cuisine)** checkbox under the **Views** section.
7. Next in the **GENERAL INFORMATION** section, leave all values as default and enter `8.x-1.0` as the **Version** field. Click on the **Download Archive** button:

The screenshot shows the 'GENERAL INFORMATION' configuration form. It includes fields for 'Name' (set to 'Recipe'), 'Description' (containing placeholder text), 'Bundle' (set to 'Default'), 'Version' (set to '8.x-1.0'), and 'Allow conflicts' (unchecked). At the bottom are 'Download Archive' and 'Write' buttons, with a red arrow pointing to the 'Download Archive' button.

- 8.** We can see that `Recipe.tar.gz` is downloaded. Now we extract this file into our `d8dev/module` directory. Open PhpStorm and see the `.yml` files and a few directories as shown in the following screenshot:



- 9.** We can see that there are many YML files. All of these files are configuration files for views, fields, and the Recipe content type. Also there is an `info.yml` file, so we can use this as a module. We can use this module in any other environment where it has Drupal 8 installed, and modules `views_semantic` and other modules installed. Follow *Chapter 1, Setting Up a Drupal Development Environment*, to install another d8dev site.

## What just happened?

We enabled the Features module and looked into how to export the Recipe content type and its fields. We also looked into how we can export views and blocks along with the Recipe feature.

## **When to use core Configuration Management compared to Features**

In Drupal 7 core, there is no system to manage configurations and the Features module can export and import configuration data bundled as code in modules. Features was used for configuration management and deployments. Because of the following issues, the module was not really suitable:

- ◆ There is no consistency in the structure of the exported configurations
- ◆ It overrides and reverts modifications of content managers

Drupal 8 has introduced Configuration Management that can deal with such issues, but that does not mean we do not need the Features module anymore. Configuration Management was not suitable to export the bundling functionality to other environments, websites, clients, or projects. That is why we still need the Features module. Drupal 8 Features will return its bundling functionality (like a blog or image gallery) rather than just managing configurations. Also, Features allows us to pick and choose what configuration data we want to add to bundle in a simplified manner. Additionally, it is much easier to update the configurations stored in the module during development. There are also some issues with configuration management:

- ◆ Adding configurations to a module is a manual process, meaning copying/pasting of YML data.
- ◆ Modules provide only initial configuration files. If we want to change the configuration files, we need to do that through writing update hooks.
- ◆ If we uninstall a module, it does not remove all the configurations. And Drupal 8 core cannot enable a module if a configuration already exists.

My conclusion is that the Features module is still very important in Drupal 8. The module can be tagged as a developer module that aims to help Drupal developers in their daily work environment to bundle functionalities and easily import changes during development.

## Summary

In this chapter, we learned more about the Views module and saw how a good Views plugin, along with some custom CSS, enabled us to create a very appealing rotating banner component for our d8dev site.

We also learned a bit about the Drupal `README.txt` file and how a module is promoted from a sandbox project to a full project.

Finally, we learned about the Features module for exporting the Recipe content type and its fields, blocks, and views as modules to use them in other environments. We also compared the Configuration Management system with the features module by focusing on a comparative study and when to use either of the two modules.

In the next chapter, we will cover some advanced search concepts and walk you through the installation and Drupal integration of the Java-based Apache Solr search engine. We will then enhance our site with a customized faceted search built on top of the Search API module.



# 11

## Searching Your Site with the Search API Module

*In this chapter, we are going to look at a more powerful and flexible replacement for the search module that comes packaged with Drupal core. We'll walk through the installation of the Apache Solr search server and look at ways of enhancing the user's search experience.*

In this chapter, we will learn:

- ◆ How to choose a suitable search configuration for your needs
- ◆ Installing and configuring the Search API module to replace the Drupal core search
- ◆ Installing and configuring the Apache Solr search server using two different methods
- ◆ Setting up a search server and a search index using the Search API module
- ◆ Creating custom faceted search blocks to enhance the user's search experience

### The Drupal core search

Like Drupal 7, Drupal 8 core comes packaged with its own built-in search module. This provides usable search functionality by querying the Drupal database directly. The Drupal core search allows you to search for user profiles and for node content. It periodically queries the database to maintain an index of content without requiring any prior configuration.

The Drupal search may be fine for your needs if you only require basic search functionality and you don't have a large amount of site traffic. However, it has several functional limitations which include the following:

- ◆ It uses the Drupal database to carry out its queries, placing extra load on the database which may already be a performance bottleneck on your site
- ◆ It's not possible to control which nodes are indexed for the search
- ◆ The content is always indexed based on the default display mode of each node
- ◆ The search only matches whole keywords for nodes

If you mind about any of these limitations, you should switch to using the Search API module.

As well as allowing additional control over how your content is searched and indexed, the Search API module gives you the option of taking a load off the Drupal database by integrating with a third-party search engine.

If the complexity of your site or the amount of traffic it gets has increased to a point to which the performance of the database is becoming a bottleneck, you'll need to use the Search API module in conjunction with a third party search engine so that the Drupal database is left to concentrate on other things. Apache Solr is an example of a third party search engine which we'll use in this chapter.

## **The Search API module**

The Search API module ([http://www.drupal.org/project/search\\_api](http://www.drupal.org/project/search_api)) replaces the Drupal core search with a framework that can be used with different kinds of underlying search engines. The search engines exist as third parties independent of the Drupal site and its database.

Thomas Seidl (also known as "drunken monkey") is one of the main contributors to the Search API module. This is how he explained its origins:

*"Search API was created in 2010. I was involved in discussions about how to improve Drupal's core search module and turn it into more of a framework for Drupal 8. The problem was that there was no search framework in Drupal and all search-related modules had to include the same boilerplate code over and over. I took the best parts of that discussion and turned them into a contributed module and search framework for Drupal 7."*

The Search API module can be extended with other contributed modules to provide very powerful search functionality. Search API also integrates with the Views module, which allows you to create Views listing pages based on search terms entered by the user.

The Apache Solr search engine we'll use later is open source, fast, and has many features. However, in place of a search engine, we also have the option of continuing to query the Drupal database with the Search API module. This has the advantage that we can set up any complex configuration we require based on the Search API without installing a search engine in our development environment. Then, when we deploy our site to a production environment we can swap to using a search engine with minimal re-configuration.

We'll start by setting up the Search API module to search using the Drupal database. This will allow us to explore some of the features and concepts that the Search API module provides. Later on, we'll swap to using the Apache Solr search engine with Search API and explore some more powerful search functionality.

## Time for action – basic installation and configuration of the Search API module

In the following steps, we'll download, install, and configure the Search API module:

1. First, download the `search_api` module using the method that you're used to. If you have Drush installed, as explained in the *Mobile First, Responsive Design* section of *Chapter 5, Theming in Drupal 8*, you'll be able to type the following at the command line:

```
$ drush dl search_api
$ drush en search_api
```
2. Before we go further, we should follow Drupal's suggestion to uninstall the core search module which the Search API module replaces.



Search API

The default Drupal Search module is still enabled. If you are using Search API, you probably want to [uninstall](#) the Search module for performance reasons.

You'll be notified of this via a command line message if you install the module via Drush, or the message will appear on the **Status Report** page at **Reports | Status report**. Disabling the core search module will remove the search block as well as the search pages for content and users. Don't panic, we'll replace them. To install the core search module with drush, type:

```
$ drush pm-uninstall search
```

3. Search API does not do anything on its own - we need to give it a search engine to interact with. To do this, we'll use a submodule that you will have downloaded as part of `search_api` called `search_api_db`. Enable it now using the method that you're used to.

The `search_api_db` module allows the Search API to interact directly with the Drupal database. This has the advantage of being simple to set up, and on small sites where the database does not already have significant load on it, you can use this method of search on the production server.

4. The `search_api_db` module contains a very handy submodule which sets up some sensible default configuration. You can then use this configuration as the basis of your search setup. To take advantage of this, enable `search_api_db_defaults` and then disable it again. All the configuration happens when the module is initially enabled and will remain when you disable the module.
5. If you now navigate to **Configuration | Search and metadata | Search API**, you should see a screen similar to the following:

The screenshot shows the 'Search API' configuration page in the Drupal admin interface. The top navigation bar includes 'Back to site', 'Manage', 'Shortcuts', and a user icon. Below the navigation is a toolbar with links for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. The main content area is titled 'Search API' with a star icon. The URL in the browser is 'Home > Administration > Configuration > Search and metadata'. A note below the title explains that indexes are grouped by server, and a server is the definition of the actual indexing, querying and storage engine (e.g., an Apache Solr server, the database, ...). It also states that an index defines the indexed content (e.g., all content and all comments on "Article" posts). There are two buttons at the top: '+ Add server' and '+ Add index'. Below these are two rows in a table:

| TYPE   | NAME  | STATUS | OPERATIONS |
|--------|---|--------|------------|
| Server | Database Server<br>Default database server created by the Database Search Defaults module     | ✓      | Edit       |
| Index  | Default content index<br>Default content index created by the Database Search Defaults module | ✓      | Edit       |

## ***What just happened?***

You downloaded and installed the Search API module and used one of its submodules to install some default configuration. We can now see that this configuration includes a search server and a search index, and we'll examine what this means.

In the next few sections, we'll review some aspects of the default settings that have been set up by the `search_api_db_defaults` module. Note that some of these settings and screen layouts may change in subsequent versions of the modules that we're using. We won't cover all the settings, but hopefully enough to give you a good start in customizing the settings to your own needs.

## An explanation of search servers and search indexes

In Drupal terminology, a search server defines how searchable data is indexed. This includes the way in which it is queried on your Drupal site and how the search index is stored. These things are dictated by the *backend* that you choose to give your search server. In this chapter, we'll be looking at search servers that use the Drupal database as a backend and the Apache Solr search engine as a backend. However, there are other backends that you can use which will define other ways in which your data is queried and indexed.

The search index defines the indexing process. The search index is associated with a search server that you previously configured to power it. The search index settings dictate things such as the types of content that gets indexed on your site, the fields that are indexed, and various ways the data is processed before, during, and after the search.

By splitting the configuration up into a search server and a search index, it is possible to swap out the search engine that performs the work whilst maintaining the way the data is queried and indexed. This is a little akin to being able to swap the petrol engine in your car for a diesel one whilst maintaining everything else about the way it looks and feels to drive.

It's also possible to have more than one search server and choose which one is most appropriate to use, depending on the point you're at in site development or which of your environments you're working in.

To extend the search configuration even further, it is also possible to have multiple search indexes. Imagine a future in which your recipe site is so popular that you start selling cookery books of your recipes on your site. You may want to have a separate search index that contains only details of the books, so that when users type `India` in a separate search box, they only see cookery books related to Indian cooking in the search result. Both your search indexes could still be based on the search server with the Apache Solr backend.

## Search server

Having configured the Search API module in the previous steps, the first item you'll see listed on the Search API configuration table is a search server named Database Server. As the name suggests, this is a search server based on the Drupal database as the backend.

## *Searching Your Site with the Search API Module*

---

Click on **Edit** in the operations column and you will be taken to the Database Server configuration:

The screenshot shows a Drupal administrative interface for editing a search server. The top navigation bar includes links for Back to site, Manage, Shortcuts, and the user admin. Below the navigation is a toolbar with Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. The main content area has tabs for View and Edit, with Edit selected. The title is "Edit search server Database Server". The breadcrumb trail shows Home > Administration > Configuration > Search and metadata > Search API > Database Server. The form fields include:

- Server name \***: A text input field containing "Database Server". A tooltip indicates the machine name is default\_server [Edit].
- Enabled**: A checked checkbox.
- Description**: A text area containing "Default database server created by the Database Search Defaults module".
- Backend \***: A radio button group where "Database" is selected.
- Configure DATABASE BACKEND**: A collapsed section that indexes items in the database. It shows "Database default > default" and a dropdown for "Minimum word length" set to 3. A tooltip for the minimum word length is present.
- Save** and **Delete** buttons at the bottom.

The server configuration is fairly self-explanatory. A server has a name and it should be enabled via the checkbox. It also has a backend, which in this case is Database and is provided by our `search_api_db` module. You can also set the minimum number of letters in words that will be used in a search.

Next up, we'll look at the configuration for default content index.

## Search index

We now require our search index settings. Configuration for the search index is also on the Search API page at **Configuration | Search and metadata | Search API**. Click on **Edit** next to the **Default content index** to see the configuration page:

The screenshot shows the 'Edit search index Default content index' configuration page. At the top, there are tabs for View, Edit, Fields, and Processors. The 'Edit' tab is active. Below the tabs, the breadcrumb navigation shows: Home > Administration > Configuration > Search and metadata > Search API > Default content index.

**Index name \***: Default content index (Machine name: default\_index [Edit]).

**Data sources \***: Content (selected). Other options include Comment, Contact message, and Custom block.

**Server**: Database Server (selected). Other options include No server.

**Description**: Default content index created by the Database Search Defaults module.

**Index Options**: (button)

Buttons at the bottom: Save (highlighted), Delete.

You'll see that there are a number of data sources available for indexing. We're not just limited to users and node content as we were with the Drupal core search. For now we'll only index content, as it's already selected in the default configuration we installed.

You'll note also that the search index is linked to a search server by way of a radio button selection. The **Server** field is set to **Database Server** in this case, as that's the only one we have set up so far.

## Fields

Next, move on to the **Fields** tab at the top of the page. This is where we set which fields on each of the data sources actually get indexed:

The screenshot shows the 'Manage fields' interface for a search index named 'Default content index'. The top navigation bar includes links for Back to site, Manage, Shortcuts, admin, Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help. Below the navigation is a breadcrumb trail: Home > Administration > Configuration > Search and metadata > Search API > Default content index. A prominent blue button labeled '+ Add fields' is visible. The main content area is titled 'Manage fields for search index Default content index'. It features two expandable sections: 'GENERAL' and 'CONTENT'. The 'GENERAL' section contains two rows: 'Item language' (Machine Name: 'search\_api\_language', Type: 'String', Boost: '1.0') and 'Rendered item' (Machine Name: 'rendered\_item', Type: 'Fulltext', Boost: '1.0'). The 'CONTENT' section contains eight rows: 'Title' (Machine Name: 'title', Type: 'Fulltext', Boost: '8.0'), 'Authored on' (Machine Name: 'created', Type: 'Date', Boost: '1.0'), 'Changed' (Machine Name: 'changed', Type: 'Date', Boost: '1.0'), 'Publishing status' (Machine Name: 'status', Type: 'Boolean', Boost: '1.0'), 'Sticky at top of lists' (Machine Name: 'sticky', Type: 'Boolean', Boost: '1.0'), 'Tags' (Machine Name: 'field\_tags', Type: 'String', Boost: '1.0'), and 'Author name' (Machine Name: 'author', Type: 'String', Boost: '1.0'). Each row has a 'REMOVE' link on the right.

Inside the **general** table, you'll see **Item language** and **Rendered item** listed. These are properties that are common to all types of searchable data sources. The **Rendered item** field allows you to go to a full text search based on what an entity looks like when rendered in Drupal.

In the **CONTENT** table further down the page, you'll see fields that are common to nodes. For each field on the field configuration page, you can set the boost value, which controls the relative significance that a search match in that field is given when it comes to ordering the search results.

## Processors

Processors act on data or the query at different points during the search and indexing process, and make alterations. Processes have their own tab on the search index where they can be configured.

**Manage processors for search index *Default content index***

View Edit Fields **Processors**

Home » Administration » Configuration » Search and metadata » Search API » Default content index

Configure processors which will pre- and post-process data at index and search time.

**ENABLED**

- Aggregated fields  
Add customized aggregations of existing fields to the index.
- Content access  
Adds content access checks for nodes and comments.
- Highlight  
Adds a highlighted excerpt to results and highlights returned fields.
- HTML filter  
Strips HTML tags from fulltext fields and decodes HTML entities. Use this processor when indexing HTML data, e.g., node bodies for certain text formats. The processor also allows to boost (or ignore) the contents of specific elements.
- Ignore case  
Makes searches case-insensitive on selected fields.
- Ignore characters  
Configure types of characters which should be ignored for searches.
- Node status  
Exclude unpublished nodes from node indexes.
- Rendered item  
Adds an additional field containing the rendered item as it would look when viewed.
- Stopwords  
Allows you to define stopwords which will be ignored in searches. **Caution:** Only use after both 'Ignore case' and 'Tokenizer' have run.
- Tokenizer  
Splits text into individual words for searching.
- Transliteration  
Makes searches insensitive to accents and other non-ASCII characters.

## *Searching Your Site with the Search API Module*

---

Half way down this page, you see the processor order table which allows you to control the order in which the processors run:

The screenshot shows the 'PROCESSOR ORDER' configuration page. It is divided into three main sections: 'PREPROCESS INDEX', 'PREPROCESS QUERY', and 'POSTPROCESS QUERY'. Each section contains a list of processors with a 'Show row weights' link.

- PREPROCESS INDEX:**
  - Node status
  - Rendered item
  - Content access
  - Stopwords
  - Ignore case
  - Transliteration
  - HTML filter
  - Tokenizer
- PREPROCESS QUERY:**
  - Stopwords
  - Ignore case
  - Transliteration
  - HTML filter
  - Tokenizer
  - Content access
- POSTPROCESS QUERY:**
  - Stopwords
  - Highlight

Below these sections is a 'Processor settings' panel. It lists four processors with their current status:

- Highlight**: Enabled
- HTML filter**: Enabled
- Ignore case**: Enabled
- Rendered item**: Enabled

On the right side of the panel, there is a list of fields where the 'Highlight' processor can be enabled:

- Item language
- Content » Title
- Rendered item
- Content » Tags
- Content » Author name
- Node access information
- Content » Content type

For example, the **Rendered item** processor, that assembles the whole entity to be searched, has to be run before the data gets indexed. Processors such as the highlight that gets put on the results of the search to show the term you searched for happens after the query is executed, so you will note that this is in the **POSTPROCESS QUERY** section.

If you're making changes to processors, pay special attention to the **Processor settings** in the vertical tabs at the bottom of the page. This is where the exact behavior of each processor is configured. If enabled processors aren't behaving exactly as you expect them to, it's these settings which should be your first port of call.

**Processor settings**

|                                   |   |
|-----------------------------------|---|
| <b>Highlight</b><br>Enabled       | Enable this processor on the following fields       |
| <b>HTML filter</b><br>Enabled     | <input type="checkbox"/> Item language              |
| <b>Ignore case</b><br>Enabled     | <input checked="" type="checkbox"/> Content » Title |
| <b>Rendered item</b><br>Enabled   | <input checked="" type="checkbox"/> Rendered item   |
| <b>Stopwords</b><br>Enabled       | <input type="checkbox"/> Content » Tags             |
| <b>Tokenizer</b><br>Enabled       | <input type="checkbox"/> Content » Author name      |
| <b>Transliteration</b><br>Enabled | <input type="checkbox"/> Node access information    |
|                                   | <input type="checkbox"/> Content » Content type     |

**Save**

## Populating your search index

Now that we've set up our search backend and search index, we need to use them to index the content on our site.

Bulk indexing of your content can be performed by clicking on the **Index now** button at the foot of the **Default content index view** tab. You'll see a blue progress bar to show progress as your content is indexed and then you'll be returned to the content index view tab.

Content can also be indexed during a cron run. One way to trigger a cron run is with the drush command:

```
$ drush cron
```

## *Searching Your Site with the Search API Module*

---

The number of content nodes that are indexed each time cron runs is 50 by default. If you have 50 items of content or less, the blue progress bar should show 100 percent after one cron run. If you have more content, you might have to run cron a few times before it is all indexed. You can adjust the number of nodes that are indexed on each cron run (that is, the cron batch size) on the **Edit** tab of your search settings. Scroll to the bottom of the page and expand the **Index Options** where you'll be able to adjust this value:

The screenshot shows the 'Default content index' configuration page. At the top, there are tabs for View, Edit, Fields, and Processors. Below the tabs, the breadcrumb navigation shows: Home > Administration > Configuration > Search and metadata > Search API. The main content area displays the index status: 'Default content index created by the Database Search Defaults module'. A progress bar indicates '2/2 indexed' (100%). Below the progress bar, there are several configuration options:

|                     |  |
|---------------------|--|
| Status              | enabled ( <a href="#">disable</a> )  |
| Data source         | Content  |
| Tracker             | Default  |
| Server              | Database Server  |
| Server index status | There are 2 items indexed on the server for this index. ( <a href="#">More information</a> ) |
| Cron batch size     | During cron runs, 50 items will be indexed per batch.  |

At the bottom of the configuration section, there is a button labeled '▼ INDEX NOW' with three dropdown options: 'Index all' (selected), 'items in batches of 50 items', and 'Index now'. Below this, there are two links: 'Queue all items for reindexing' and 'Clear all indexed data'.

Whether you ran cron or clicked on the **Index now** button, your content should now be indexed.

You'll note that there are some links at the bottom of the content index view tab that provide some extra control over the indexation of your content:

- ◆ **Queue all items for reindexing** marks all the currently indexed items of content as ready to be indexed again the next time cron runs or you click on **Index now**. Until content re-indexing is triggered, the current search index will continue to be used. This helps deliver uninterrupted searching for your users, even when content must be re-indexed.

- ◆ **Clear all indexed data** makes the current search index immediately unavailable and searches will stop yielding results until content is re-indexed.

## Exposing the search to users

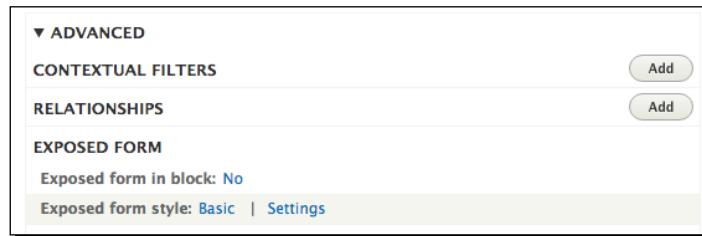
So now we have a search index with content in it, but how do we actually allow the end users to perform a search on our Drupal site?

One of the features of Search API is that it integrates well with the Views module. This means that you can search content using the search server you set up and display it in a view. The `search_api_db_defaults` module that you used earlier provides some views configuration which implements the search index and search server that was configured.

To see this View, go to `/search/content` on your site. Try searching the content for a certain word and check that you get sensible results. You should find that if you have two items of content, one of which has your search keyword in the title and one in the body text, that the search results are ordered according to the boost values that you set in the **Fields** tab of your search index settings section earlier. By default, the title field has a boost value of eight while the rendered item has a boost value of one. Therefore, content that has your keyword in its title should appear before content that has your keyword in its body or elsewhere.

In order to better replicate the Drupal core search functionality that we disabled earlier in favor of the Search API, it would be nice to have a search field that we can display site-wide so that users don't need to navigate to `/search/content` to perform a search.

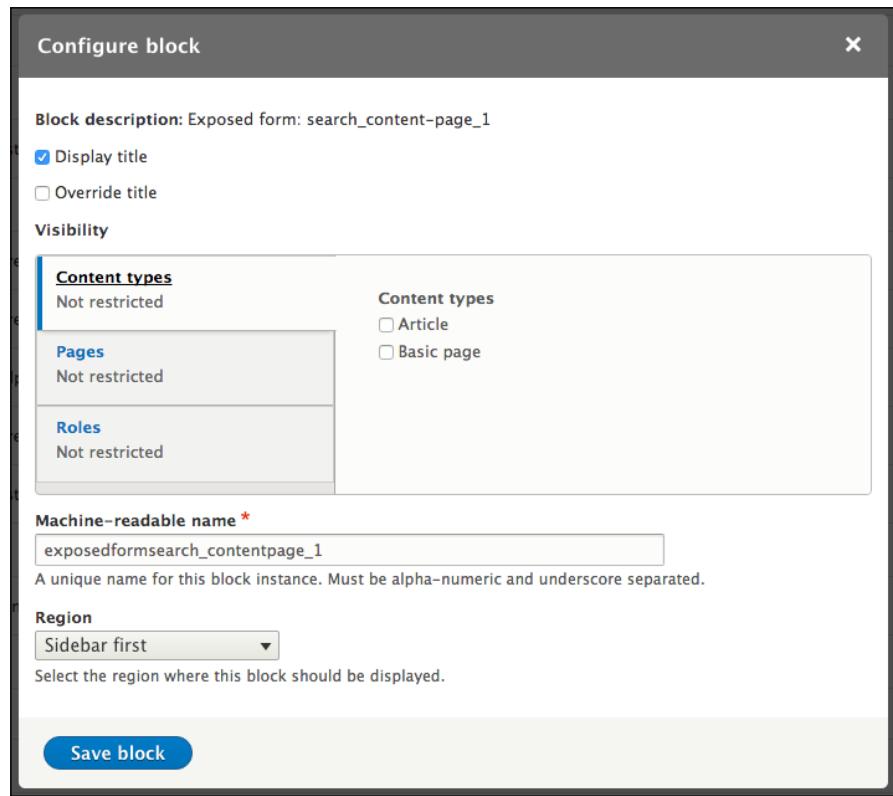
To achieve this, we'll expose the search form in a block. Navigate to **Structure | Views** and click on the **Edit** button next to the **Search content** view. Expand the **Advanced** section and toggle **Exposed form in block** from **No** to **Yes**. Click on the **Save** button:



## *Searching Your Site with the Search API Module*

---

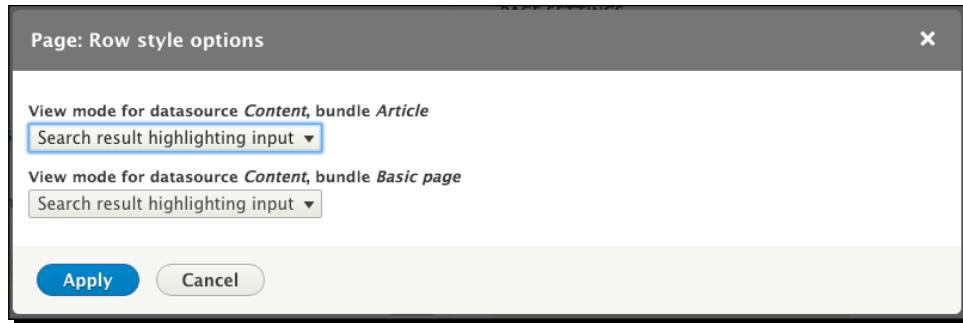
The exposed search field that appears at the top of our new search results page is now available in a block that we can configure to appear on pages on our site. Go to **Structure | Block layout** and click on **Place block** next to the region you want to place it in. Somewhere in the list that appears, you will see the block **Exposed form: search\_content-page\_1**. Configure the block settings in the usual way:



You should now have a sitewide search block that replaces the core search block that we removed.

## **Altering the search display**

One advantage of Search API is that the way each item in your search results is displayed can be controlled with your content type display settings. You will note that the search content view uses a node display mode of **Search result highlighting input**.



This is a node display mode which is supplied by **search\_api\_db\_defaults**. You can change the settings of this display mode from the settings of each content type.

Go to **Structure | Content types**, choosing the content type whose display settings you'd like to change, and then go to tab **Manage display | Search result highlighting input**:

| FIELD           | LABEL      | FORMAT             |   |
|-----------------|------------|--------------------|---|
| ⊕ Image         | - Hidden - | Image              | Image style: Thumbnail (100x100)<br>Linked to content |
| ⊕ Body          | - Hidden - | Summary or trimmed | Trimmed limit: 300 characters                         |
| ⊕ Links         |            | Visible            |   |
| <b>Disabled</b> |            |                    |   |
| ⊕ Tags          | Above      | - Hidden -         |   |
| ⊕ Comments      | Above      | - Hidden -         |   |

Try changing the fields from your content type that are displayed in the search results. For example, try hiding the image and setting the **Body format** to **Trimmed** with a trimmed limit of 400 characters.

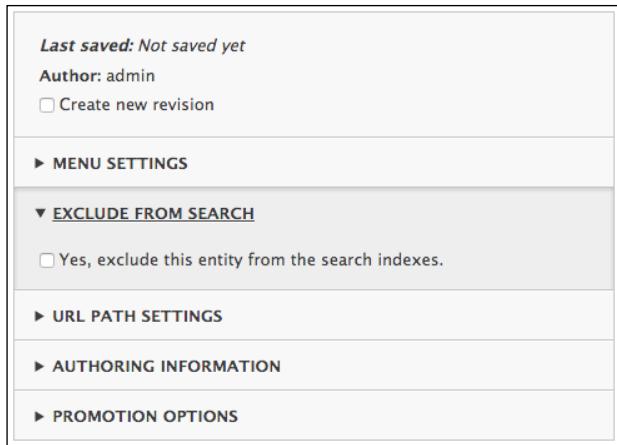
You should now get search results as something similar to the following screenshot:



## Excluding entities from being indexed

With the configuration we've set up so far, all the nodes on your site will be indexed and will be searchable. What if we'd like to choose whether individual recipes we create on our site should appear in the search results or not. Perhaps we'd like to create promotional links to special recipes that are not available through a site search, for example.

Well, there's a contributed module for that called Search API Exclude Entity ([https://www.drupal.org/project/search\\_api\\_exclude\\_entity](https://www.drupal.org/project/search_api_exclude_entity)). When you download and install it, you'll find that there is an additional field type available that can be added to your nodes and other entities. If you add this field to your recipe content type, then when you next create or edit a recipe, you'll note that there are some extra metadata options that appear on the recipe edit screen:



Next, go to the **Processors** configuration on your search index: [Configuration | Search and metadata | Search API](#). Click on **Edit** next to the **Default content index** to see the configuration page, then choose the **Processors** tab. You'll see that you have a new Search API Exclude Entity that can be enabled. Once you've enabled this processor, scroll down to the **Processor settings** section at the bottom of the screen and check the checkbox next to the field name that you just added to your content type.

Try creating a new recipe and excluding it from the search. Check that all your content has been indexed and then do a search for your new recipe. You should find that it doesn't appear in the search results.

## Installing Apache Solr as the search backend

The main reason for using Search API over the core search module is that we can plug in different search engines and use them to index your content. This really becomes an advantage when we start working with a large site that contains a lot of content.

Apache Solr is an open source search engine used by many high profile enterprise level sites. It is not part of Drupal or a Drupal module and is very often used independently of Drupal. However, it integrates well with the Drupal Search API module, as we'll see.

Solr has the following advantages over using the Search API with the database:

- ◆ It's faster
- ◆ It does not rely on your Drupal database, freeing up database resources for Drupal to do the rest of its work with

- ◆ It can be hosted on a server separate from your database and your code base which is useful when you need to scale your infrastructure
- ◆ You can search for phrases and use wildcards

We'll examine these same defaults later in this chapter. Meanwhile, if you'd like more information on the Apache Solr project, see its website at <http://lucene.apache.org/solr/>.

There are a few different ways to install Solr. Here, we'll explain two methods, one using a virtual machine and a puppet script, and one more manual method using an Ubuntu 14.04 server.

If you follow either of these methods, Apache Solr does not block unauthorized users from accessing your Solr server by default. If you are setting up a Solr server outside your local machine, or that can be connected to over the Internet, it is extremely important that you employ some method of securing Apache Solr. Otherwise, unauthorized users will be able to make search queries and possibly even compromise your website in other ways. See the section *Securing Apache Solr with Uncomplicated Firewall* in this chapter to learn how to secure Solr.



## Installing Solr 4.x on a virtual machine with Vagrant and Puppet

This is a quick and easy method of setting up a disposable virtual machine that you can use locally for development purposes. It will behave exactly the same as a Solr instance on a real server that you can go on to set up later if you want.

A useful online service for generating Puppet deployment configuration was introduced in *Chapter 1, Setting Up a Drupal Development Environment*. This tool is called PuPHPet—Puppet with a capital PHP in the middle to denote that it is primarily intended for PHP web development. At the time of writing, the latest version of Apache Solr to be available on PuPHPet (<https://puphpet.com/>) was 4.10.\*. Therefore, these instructions will assume that you're using the 4.x version of Solr.

If you really need the latest and greatest version of Solr, you can skip to the next section where we'll explain how to install Solr manually on an Ubuntu 14.04 server.

## Time for action – creating and configuring your virtual machine

The following steps will use an online service to create a Puppet script that you can use with Vagrant to launch a virtual machine locally. Puppet, Vagrant, and the PuPHPet site were introduced in *Chapter 1, Setting Up a Drupal Development Environment* of this book. If you haven't already got Puppet and Vagrant set up locally, follow the instructions there to get them working.

1. Create your VM at PuPHPet. Point your browser at <https://puphpet.com> and click the big green **Get started right away** button:

The screenshot shows the PuPHPet website interface. On the left is a sidebar with a tree view of deployment targets: Deploy Target (selected), System, Web Servers, Languages, Databases, Mail Tools, Work Queues, Search Servers, and Create Archive. The main content area has several sections: "Welcome to PuPHPet" (a simple GUI for setting up virtual machines for web development), "Easily share with friends and coworkers" (using YAML files and ZIP archives), "Deploy to any server in the world!" (support for Rackspace, Digital Ocean, Linode, and more), and "Open sourced, MIT licensed." (with a link to help out). There are also "Important Notices" (mentioning multi-machine support and minimum Vagrant version 1.8.x) and "Latest news" (listing items like Multi-Machine Support Added, Symlink Support in Windows and VirtualBox, Better SSL Security, PHP7 support added, and Drag & drop BC breaking changes). At the bottom is a large green button with the text "Get started right away. It's free! ⌂".

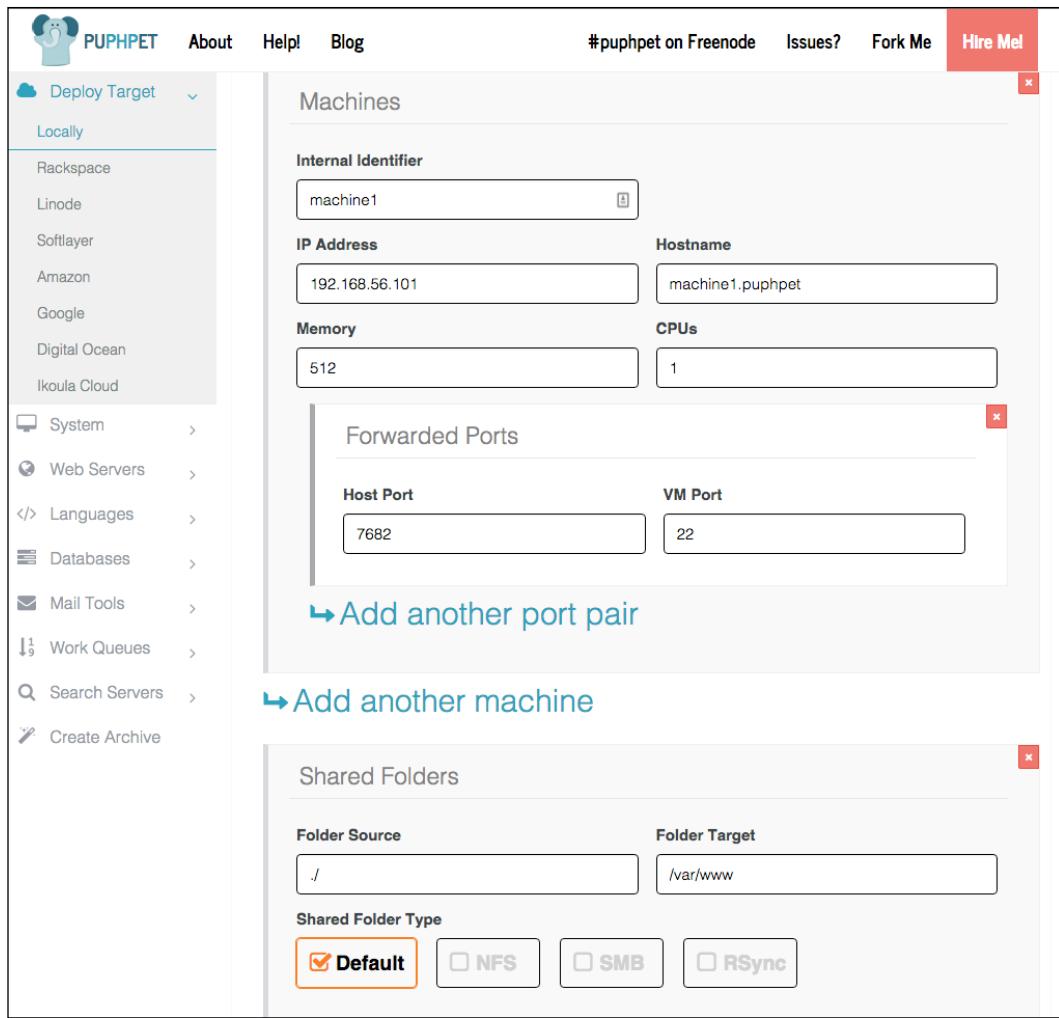
During the wizard that you'll be guided through, you can generally accept the default options. Just make sure that when you get to the **Search Servers** section you choose to install the latest 4.\* version of Apache Solr. We'll also glance at a couple of other options here for creating a nice, clean virtual machine that you can use time and time again.

## *Searching Your Site with the Search API Module*

On the first page of the wizard after you've clicked on the **Get started** button, make sure you have **Deploy to Local Host** checked, along with VirtualBox as the provider. We'll also select Ubuntu Trusty 14.04 LTS x64 as the Distro:

The screenshot shows the PUPHPET interface for setting up a local virtual machine. On the left, a sidebar lists various deployment targets like Rackspace, Linode, Softlayer, Amazon, Google, Digital Ocean, and Ikoula Cloud. Below that is a list of system components: System, Web Servers, Languages, Databases, Mail Tools, Work Queues, Search Servers, and Create Archive. The main content area is titled "Local VM Details" and contains a sub-section "Deploy to Local Host" with a checked checkbox. A note below it says, "The virtual machine will live inside your local PC. [Click here for instructions.](#)" Under the "Provider" section, "VirtualBox" is selected with a checked checkbox. Other providers listed are VMWare Fusion (unchecked), VMWare Workstation (unchecked), and Parallels Desktop (unchecked). In the "Distro" section, "Ubuntu Trusty 14.04 LTS x64" is selected with a checked checkbox. Other distros listed are CentOS 6 x64 (unchecked), Debian Wheezy 7 x64 (unchecked), Ubuntu Trusty 14.04 LTS x32 (unchecked), and Ubuntu Precise 12.04 LTS x64 (unchecked).

If you scroll further down this page, you'll find options for setting an **Internal Identifier**, **Hostname**, and **IP Address**. You may want to change these if you have other virtual machines already running and need to distinguish this one from the others.

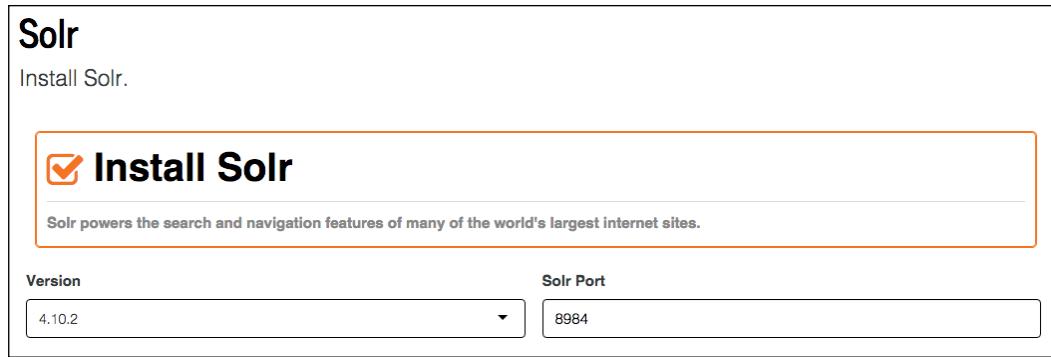


It's also important that you have a folder that will be shared between your physical host machine and the virtual machine you are creating. By default, this is configured for you.

Follow the wizard using the big green buttons at the bottom of the screen and accept the defaults for the rest of the **System** section.

When you get to the **Web Servers** section, you can deselect both Apache and Nginx (Apache Solr doesn't require either of them). All the options can also be deselected in the **Languages** and **Databases** sections—Solr does not require any of these.

Keep moving through the wizard, continuing to accept the further defaults until you get to **Search Servers**. Choose **Apache Solr** and select the latest version from the drop-down box:



2. Download and set up your virtual machine. Now you can hit the **Create Archive** button to download a ZIP file containing the Puppet configuration you've just set up.

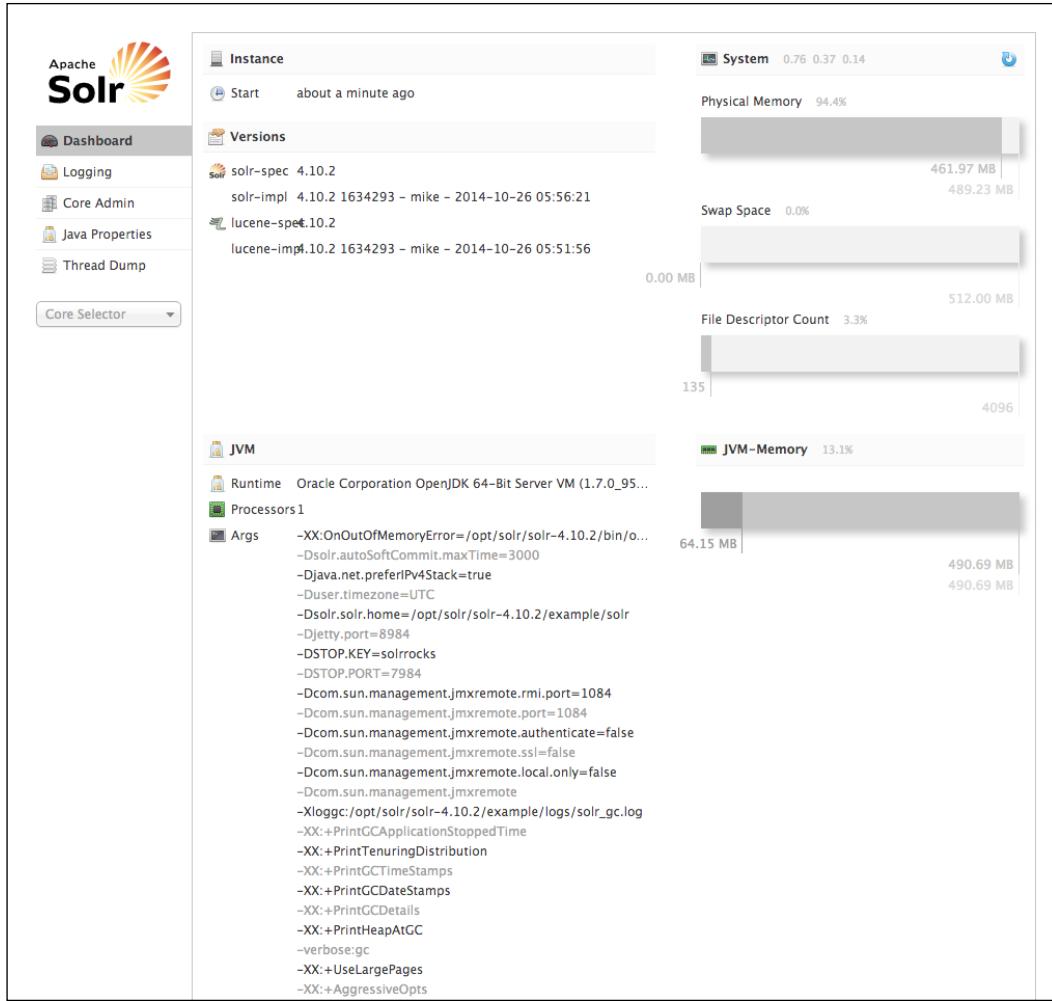
As previously explained in *Chapter 1, Setting Up a Drupal Development Environment*, you can now run your virtual machine by extracting the ZIP archive, navigating to the extracted folder at your command line (which will contain a file called `Vagrantfile`), and typing:

```
$ vagrant up
```

It will take a little while the first time you start the virtual machine because it has to download a large disk image containing the operating system. During this process you should get some verbose output at your command line, informing you of what is happening.

If everything ran correctly, you should now be able to open a browser and navigate to `http://localhost:8983/solr`. If this doesn't work, check that the port number is correct by scrolling back through the output at your command line and checking which local port has been forwarded to 8983, the Apache Solr port.

Here's what you should now see in your browser:



3. Install the Drupal schema. Apache Solr is now installed on your virtual machine and you can connect to it through your browser. Congratulations!

It's not going to be of much use though unless we give it some content to index and search. We need to connect it to our Drupal site. The first step to doing this is to install the correct configuration files that tell Solr how to index Drupal content.

Download and extract the `search_api_solr` module for Drupal 8. Put it in the `/modules` folder of your site, but don't enable it just yet. The configuration files required for the Vagrant Solr instance can be found in the Drupal `search_api_solr` module at `search_api_solr/solr-conf/4.x`.

To copy these files to your virtual machine, copy the `4.x` folder into the `puphpet` folder, which sits in the same folder as the `Vagrantfile` that you extracted earlier.

Next, SSH into your virtual machine by navigating to the folder that contains `Vagrantfile` and typing:

```
$ vagrant ssh
```

You will be logged in to your virtual machine over SSH, just as though it was a remote server.

The `puphpet` folder you copied the configuration into is shared between the physical host machine and the virtual machine. You can find the `4.x` folder at `/var/www/puphpet/4.x` on the virtual machine. First, back up the original Solr configuration and then move the new configuration into your Solr instance at `/opt/solr/solr-4.10.2/example/solr/collection1/conf`:

```
$ sudo cp -r /opt/solr/solr-4.10.2/example/solr/collection1/conf /  
opt/solr/solr-4.10.2/example/solr/collection1/conf-orig  
$ mv /var/www/puphpet/4.x/* /opt/solr/solr-4.10.2/example/solr/  
collection1/conf/
```

Now we'll restart Solr. An easy way to do this is just to restart our entire virtual machine.

Next, exit the SSH terminal:

```
$ exit
```

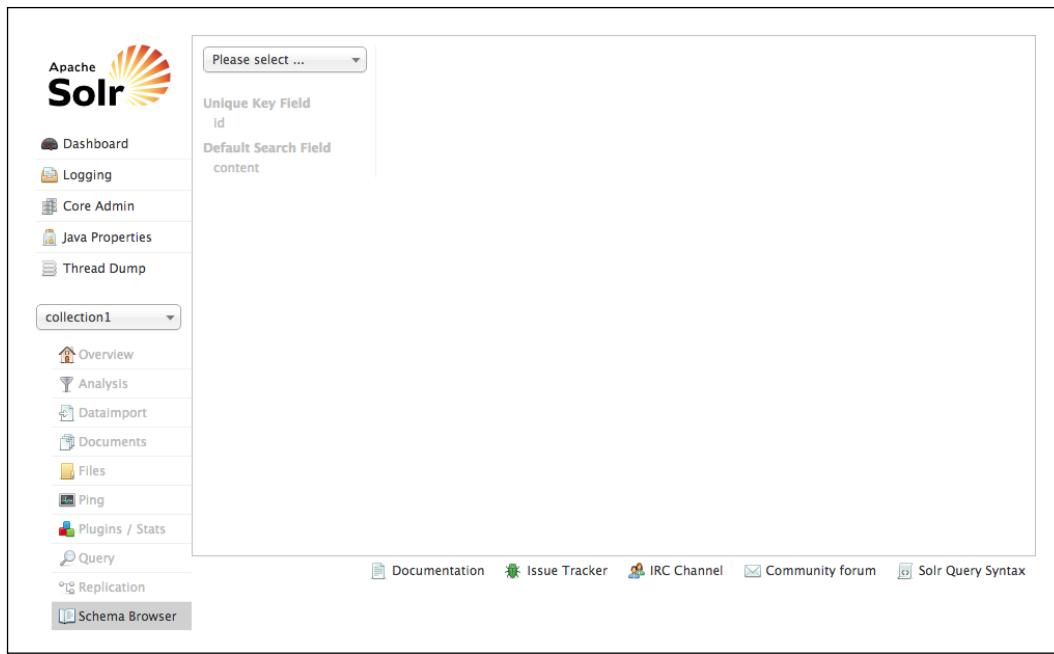
Then stop the virtual machine:

```
$ vagrant halt
```

Then start it up again:

```
$ vagrant up
```

Check that everything worked correctly by visiting the Solr admin page again in your browser.



Select **collection1** from the dropdown in the sidebar and then click on **Schema Browser** at the bottom of the sidebar. In the dropdown that then appears in the main content area of the page, you should see a bunch of Drupal related **fields**, **bundle\_name**, **entity\_id**, **entity\_type**, and so on.

Next, you'll need to enable and configure the `search_api_solr` module in Drupal. To do this now, skip to the section. The Search API Solr module.

## **What just happened?**

You used Vagrant and Puppet to set up a virtual machine running Apache Solr. You then configured your Solr server to work with Drupal by installing the configuration file that comes with the Drupal Search API Solr module. This virtual machine configuration can be reused again and again whilst you're experimenting and developing with Apache Solr.

## **Installing Solr 5.x manually on Ubuntu 14.04**

This is the method to use if any of the following apply:

- ◆ You want a later version of Apache Solr than is available on PuPHPet
- ◆ You need Solr to run on a server outside your local machine
- ◆ You want more control over the installation process

These instructions will assume that you are installing Solr 5.x on an Ubuntu 4.x box. This could be the same server that your site is running on, or one dedicated to Solr.

The version of Solr that is available in the Ubuntu package repository is one major version before the current one. Rather than simply using Ubuntu's apt package manager tool to install the earlier version, we will follow a method that allows us to install the latest 5.x version of Solr.

## Time for action – installing and configuring Solr on Ubuntu

First, we'll install some required packages and then we'll manually install the latest version of Solr from the Apache Solr site:

1. Install required packages. First, use apt-get to ensure you have the `python-software-properties` package available. This will allow you to add a new package repository:

```
$ sudo apt-get install python-software-properties
```

Now, add the unofficial Java installer repository:

```
$ sudo add-apt-repository ppa:webupd8team/java
```

Confirm that you want to add this repository when prompted and then update the list of source packages:

```
$ sudo apt-get update
```

We can now install the Oracle Java JDK Version 8:

```
$ sudo apt-get install oracle-java8-installer
```

This is an Oracle package and you will have to agree to the license terms in order to proceed. Just press *Enter* when prompted.

2. Run the Apache Solr installer. Now that we have the Java JDK installed, we can install the latest version of Apache Solr. Go to the list of mirrors available to download Solr from <http://www.apache.org/dyn/closer.lua/lucene/solr/>.

Choose a mirror and then choose the latest directory with a version number that starts with 5.\*.

Next, right-click the latest .tgz file whose name does not contain `src` and copy the link to it. At the time of writing, this was `solr-5.5.0.tgz`, where the version number appears in the commands below, replace with the version you have downloaded.

Back at the command line of the server you are installing Solr on, download the file to your home folder and untar it:

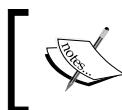
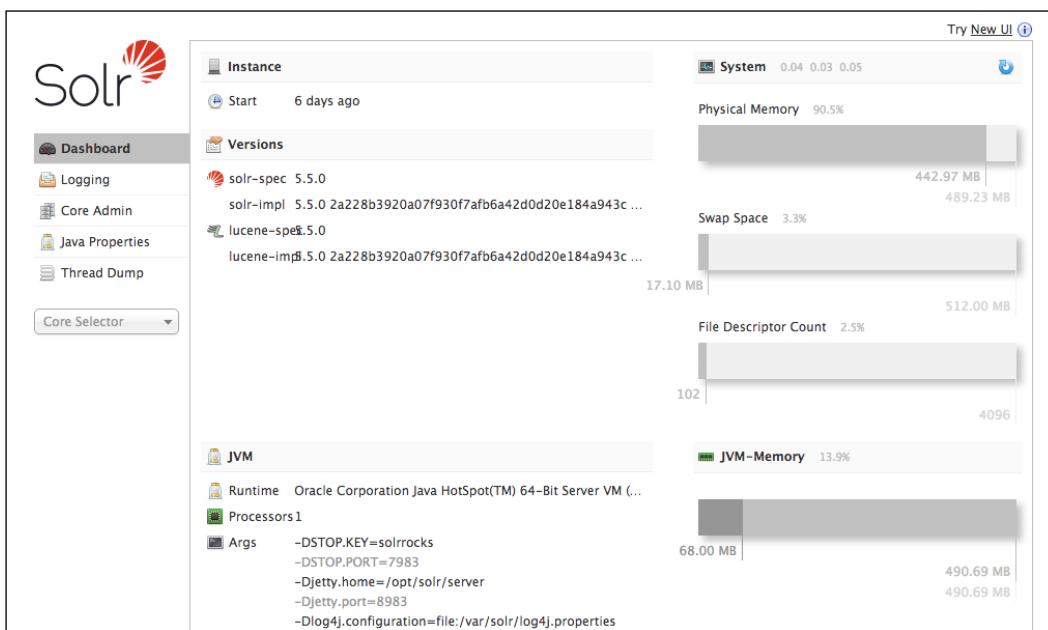
```
$ cd ~
$ wget http://<download_path>/solr-5.5.0.tgz
$ tar -xvzf solr-5.5.0.tgz
```

Solr 5.x comes with its own installer script which makes it easier to set up than the previous versions. We'll run the script as follows:

```
$ sudo bash solr-5.5.0/bin/install_solr_service.sh solr-5.5.0.tgz
```

When it starts up, Solr will output the port number that it is running on, usually 8983. Open a browser window and navigate to the IP address of your server followed by the port number. For example: <http://<Server IP address>:8983>.

You should see the Solr admin screen as follows:



Ensure you pay special attention to the following section in this chapter *Securing Apache Solr with Uncomplicated Firewall* to ensure there is no unauthorized access to your Solr interface.

- 3.** Install the Drupal schema. Apache Solr is now installed on your virtual machine and you can connect to it through your browser. Congratulations!

It's not going to be of much use though unless we give it some content to index and search. We need to connect it to our Drupal site. The first step to doing this is to install the correct configuration files that tell Solr how to index Drupal content.

The configuration files required to make Solr work with Drupal can be found in the `search_api_solr` module. Navigate to the module page on Drupal to get the link to the latest packaged `tar.gz` file. Download it to your Apache Solr server and extract the files:

```
$ cd ~  
$ wget https://ftp.drupal.org/files/projects/search_api_solr-8.x-  
1.0-alpha2.tar.gz  
$ tar -xvf search_api_solr-8.x-1.0-alpha2.tar.gz
```

You will have to replace the filenames in the preceding commands with the filename of the latest version of the module.

Next, create the directory that you will copy the Solr configuration files into:

```
$ sudo mkdir -p /var/solr/data/drupal/conf
```

Copy the files from the Drupal module into the Solr configuration directory:

```
$ sudo cp search_api_solr/solr-conf/5.x/* /var/solr/data/drupal/  
conf/
```

Set the correct ownership on the files:

```
$ sudo chown -R solr:solr /var/solr /opt/solr
```

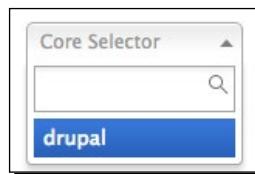
Create a new Solr core:

```
$ sudo /opt/solr/bin/solr create -c drupal
```

Restart the Solr service:

```
$ sudo service solr restart
```

Navigate again to the Solr admin screen in your browser. In the **Core Selector** drop down in the sidebar, you should now have a core called **drupal**:



## ***What just happened?***

You installed the latest version of Apache Solr on Ubuntu using the package manager. This is the process you'll use when setting Solr up from scratch on a production server. Just make sure you secure it too!

## Securing Apache Solr with Uncomplicated Firewall

It's very important that you secure Apache Solr to prevent outside access to it. If your Solr server is accessible over the Internet or even over your local network, you should employ a method of controlling unauthorized access to it.

### Time for action – configuring Uncomplicated Firewall

On Ubuntu, you can use the built in Uncomplicated Firewall tool to set access to the server at a firewall level:

1. Start by denying all access:

```
$ sudo ufw default deny
```

However, we don't want to break the SSH connection that we're currently using, so make sure that we allow access for SSH connections:

```
$ sudo ufw allow ssh
```

Also, we want to allow all connections from our local IP address:

```
$ sudo ufw allow from <ip address>
```

Here, `<ip address>` is the IP address of the machine you are connecting from. You will also want to add the IP address of your Drupal server here if it's different from your Solr server. Just run the preceding command again, changing the IP address to that of your Drupal server. For further details, see the Ubuntu UFW documentation at <https://help.ubuntu.com/community/UFW>.

2. Now enable the firewall with the following settings:

```
$ sudo ufw enable
```

You may receive a warning to say that the existing SSH commands may be disrupted. As long as you've allowed access for SSH connections with the command we ran earlier, you can proceed.

3. Now check the status of your firewall:

```
$ sudo ufw status
```

4. You should receive an output that looks as follows:

```
Status: active
```

| To | Action | From     |
|----|--------|----------|
| -- | -----  | ----     |
| 22 | ALLOW  | Anywhere |

|          |       |               |
|----------|-------|---------------|
| Anywhere | ALLOW | <ip address>  |
| 22 (v6)  | ALLOW | Anywhere (v6) |

You should now be able to access the Solr admin interface from your local machine or network but not from anywhere else. Try navigating to the following URL: `http://<Server IP address>:8983.`

## **What just happened?**

You configured Ubuntu's Uncomplicated Firewall to allow access to your Solr installation only from a specific IP address.

## **The Search API Solr module**

Prior to Drupal 8, Drupal used a module that was independent of Search API to integrate with Apache Solr. This module was called Apache Solr Search. However, it was decided that the Search API would become the module on which to base all search integrations for Drupal 8. A main contributor to both the Search API project and the earlier Apache Solr Search module for Drupal 7 is Nick Veenhof (also known as "Nick\_vh" on Drupal.org). He explained the rationale behind making the Search API module a requirement of the new Search API Solr module:

*"Drupal 8 has also helped in forcing us to rethink certain concepts and also live up to our promise to avoid the divide we had in Drupal 7 between the Apache Solr Module and Search API. Search API will be the sole provider of Apache Solr integration in Drupal 8. Search API in Drupal 8 ships with sane defaults (which you can enable or disable) that pre-configures the module so it can be used instantly."*

## **Time for action – configuring Drupal to use Apache Solr**

However you set up Apache Solr, you'll now want to configure Drupal to use your new Solr server as your search server in Drupal:

1. Installing the Search API Solr module. First we need to install the `search_api_solr` module on our Drupal site. Download the module and extract it into the `/modules` folder of your Drupal site as usual. The module has some dependencies that are managed by composer, so navigate into the directory of your module and run:

```
$ composer install
```

At the time of writing, the development version of `search_api_solr` was required in order to fix some errors on the official alpha release displayed for download on the module page. To fetch this, it is best to use Git:

```
$ git clone --branch 8.x-1.x https://git.drupal.org/project/
search_api_solr.git
```

- 2.** Configuring the Search API Solr module. When we installed the `search_api_db_defaults` module earlier on in this chapter, it created a search server based on the database for us.

Now that we have the Search API Solr module installed, we can set up a second search server based on the Solr server.

Navigate to **Configuration | Search and metadata | Search API** and click on the **Add Server** button.

Fill in the form as shown next. The Solr path should be `/solr/` followed by the name of your core. This will be `drupal` if you followed the previous instructions to install Solr 5.x or `collection1` if you followed the instructions for installing Solr 4.x:

The screenshot shows the 'Add search server' configuration page in the Drupal administration interface. The top navigation bar includes links for Back to site, Manage, Shortcuts, and the current user (admin). Below the navigation is a horizontal menu with links for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help.

The main content area has a title 'Add search server' with a star icon. The URL in the browser is 'Home > Administration > Configuration > Search and metadata > Search API'. The form fields are as follows:

- Server name \***: A text input field containing 'Solr'. To the right, a note says 'Machine name: solr [Edit]'. Below the input field is a placeholder text: 'Enter the displayed name for the server.'
- Enabled**: A checked checkbox with the note 'Only enabled servers can index items or execute searches.'
- Description**: A large text area with a placeholder text: 'Enter a description for the server.'
- Backend \***: A radio button group where the 'Solr' option is selected. The note below says 'Choose a backend to use for this server.'
- CONFIGURE SOLR BACKEND**: A section header. Below it, a note says 'Please configure the used backend.' A yellow warning box contains the note 'Index items using an Apache Solr search server.' and a dropdown menu labeled 'HTTP protocol' with 'http' selected. The note below the dropdown says 'The HTTP protocol to use for sending queries.'
- Solr host \***: A text input field containing 'localhost'. The note below says 'The host name or IP of your Solr server, e.g. localhost or www.example.com.'
- Solr port \***: A text input field containing '8983'. The note below says 'The jetty example server is at port 8983, while Tomcat uses 8080 by default.'

## *Searching Your Site with the Search API Module*

---

If you've secured the Solr admin interface using the firewall rules as explained in the previous section, you will not need to add login details for HTTP authentication. Leave this blank and click on **Save**:

The screenshot shows the Solr configuration page within the Drupal administration interface. At the top, there are tabs for Back to site, Manage, Shortcuts, and admin. Below the tabs, the Solr path is set to /solr. The main content area is divided into sections: **BASIC HTTP AUTHENTICATION**, **ADVANCED**, and **MULTI-SITE COMPATIBILITY**. In the **BASIC HTTP AUTHENTICATION** section, there are fields for Username and Password. A note states: "If this field is left blank and the HTTP username is filled out, the current password will not be changed. If your Solr server is protected by basic HTTP authentication, enter the login data here." In the **ADVANCED** section, there are several checkboxes: "Return an excerpt for all results", "Retrieve result data from Solr", "Highlight retrieved data", and "Skip schema verification". There is also a dropdown for "Solr version override" set to "Determine automatically". In the **MULTI-SITE COMPATIBILITY** section, there is a checkbox for "Retrieve results for this site only".

Now that you've created a second search server in the Drupal configuration, one of the advantages of the Search API module is that you can swap between them easily. You may want to dedicate your Solr server to the production site for example, but you still want to be able to perform searches for development purposes on your staging and development sites.

In development and staging environments, you can set your search index to use the database, and on the production environment, you can swap it to use Solr.

Edit the default content index that you set up earlier in this chapter and change the **Server** radio button to **Solr**:

The screenshot shows the 'Edit search index Default content index' page in the Drupal Admin interface. The 'Edit' tab is selected. Under 'Data sources', 'Content' is chosen. Under 'Server', 'Solr' is selected. The 'Enabled' checkbox is checked. The 'Description' field contains 'Default content index created by the Database Search Defaults module'. At the bottom are 'Save' and 'Delete' buttons.

Click on **Save** and then re-index your content. That's it! Go back to your search page and test out the new search, this time powered by Solr.

## **What just happened?**

You installed Drupal's Search API Solr module and configured it to connect to your Apache Solr installation via a new search server. You then connected your search index to the new search server and indexed your content using Solr.

## Using the read-only mode

In some situations, you may want to make your search index read-only. This can help if you implement a Drupal development workflow across several servers. For example, when you copy the Drupal database down from production to your staging and development environments in order to keep them updated with the latest content from your production site. Rather than create a new Apache Solr server for each of your environments, you'll want to use the same Apache Solr server but ensure that only content added to your production server gets indexed.

On your staging and production environments, you can check the **Read only** checkbox in the **Index Options**. This will prevent any test content you create here making it into the Apache Solr index:

The screenshot shows a configuration interface for an index. At the top, there's a section titled "▼ INDEX OPTIONS". Inside this section, there are two checkboxes: "Read only" (which is checked) and "Index items immediately" (which is also checked). Below these checkboxes, there's a "Cron batch size" input field containing the value "50". A note below the input field states: "Set how many items will be indexed at once when indexing items during a cron run. "0" means that no items will be indexed by cron for this index, "-1" means that cron should index all items at once." At the bottom of the configuration area, there are two buttons: "Save" and "Save and edit".

You should note that if you add a new content type to the system, you will need to ensure that the search index is not set to read only in order for the content type to be indexed. When the index is set to read-write (that is, the read only checkbox is unchecked), any new content created using your new content type will be automatically indexed as usual.

## Search facets

To further showcase the power of the Search API, we're going to create a facet block to accompany the search listing. You may be familiar with facet blocks from other websites that have a lot of categorized content to search and sort.

For example, here are some faceted search blocks from a popular recipe website:



<http://www.bbcgoodfood.com/search/recipes?query=family>

## Time for action – building faceted search blocks

Let's look at how you can build one of these faceted search blocks using the Search API and the Drupal 8 facets module:

1. Download and enable the facets module from (<https://www.drupal.org/project/facets>) in the way you're used to.
2. Navigate to **Configuration | Search API | Facets** and click on the blue Add Facet button. As an example, we'll create a facet block that uses words from the titles of our recipes. However, if you have categorized your recipes using taxonomy terms, you could also base your facet block on that field.

Give the facet a human readable name and a machine name:

The screenshot shows the 'Add facet' configuration page in the Drupal Admin interface. The top navigation bar includes links for Back to site, Manage, Shortcuts, and admin. Below the navigation is a toolbar with links for Content, Structure, Appearance, Extend, Configuration, People, Reports, and Help.

The main content area has a title 'Add facet' with a star icon. The path 'Home > Administration > Configuration > Search and metadata > Facets' is displayed above the form.

The form fields are as follows:

- Facet name \***: A text input field containing 'Title'. To its right is a link 'Machine name: title [Edit]'.
- The name of the facet for usage in URLs \***: A text input field containing 'title'. A note below it says 'A unique machine-readable name. Can only contain lowercase letters, numbers, and underscores.'
- Facet source \***: A dropdown menu set to 'Search API view: Search content, display: Page'.
- Facet field \***: A dropdown menu set to 'Title'.
- The weight of the facet \***: A text input field containing '0'.
- Enabled**: A checked checkbox with the note 'Only enabled facets can be displayed.'

A blue 'Save' button is located at the bottom left of the form.

3. The facet block needs a source to get its data from. In the **Facet field** dropdown, you'll have one option which will be the search view that we created earlier. Set the **Facet field** to **Title** and the weight of the facet to 0, then save the settings.

- 4.** You can also configure the display settings of your facet by clicking the **Display** tab at the top of the page. Here, you can choose options such as whether to display your facet list as a dropdown, checkboxes, or list of links. You can also choose the way the facet list behaves, such as hiding items that will not narrow the currently displayed search results down any further. We're going to aim to make our facet list similar to the example at the start of this section. Therefore, we'll choose **List of links** as the **Widget** and we'll check **Hide non narrowing results**:

Manage processors for facet *Title* ☆

[Edit](#) [Display](#)

Home » Administration » Configuration » Search and metadata » Facets

**Widget \***

- List of checkboxes
- Dropdown
- List of links

The widget used for displaying this facet.

**LIST OF LINKS SETTINGS**

- Show the amount of results

**FACET SETTINGS**

- Count limit  
Show or hide facets with based on item count.
- Exclude specified items  
Excludes items by node id or title.
- Hide active items  
Do not display items that are active.
- Hide non narrowing results  
Do not display items that do not narrow results.
- List item label  
Display the list item label instead of the key
- Transform uid to username  
Show the username instead, when the source field is a user id.
- Translate taxonomy terms  
Translate the taxonomy terms
- URL handler  
Triggers the URL processor, which is set in the Facet source configuration.
- Hide facet when facet source is not rendered  
When checked, this facet will only be rendered when the facet source is rendered. If you want to show facets on other pages too, you need to uncheck

## *Searching Your Site with the Search API Module*

---

- 5.** You have now created a facet which is related to a facet source. The facets table you now see at **Configuration | Search API | Facets** should look as follows:

The screenshot shows the Drupal admin interface with the title 'Facets'. The URL is 'Home > Administration > Configuration > Search and metadata'. A success message in a green box says 'The facet Type has been deleted.' Below is a table with columns: TYPE, TITLE, ENABLED, and OPERATIONS. There are two rows: one for 'Facet source' (search\_api\_views:search\_content:page\_1) with 'Configure' in OPERATIONS, and one for 'Facet' (Title: title - checkbox) with 'Edit' and a dropdown in OPERATIONS.

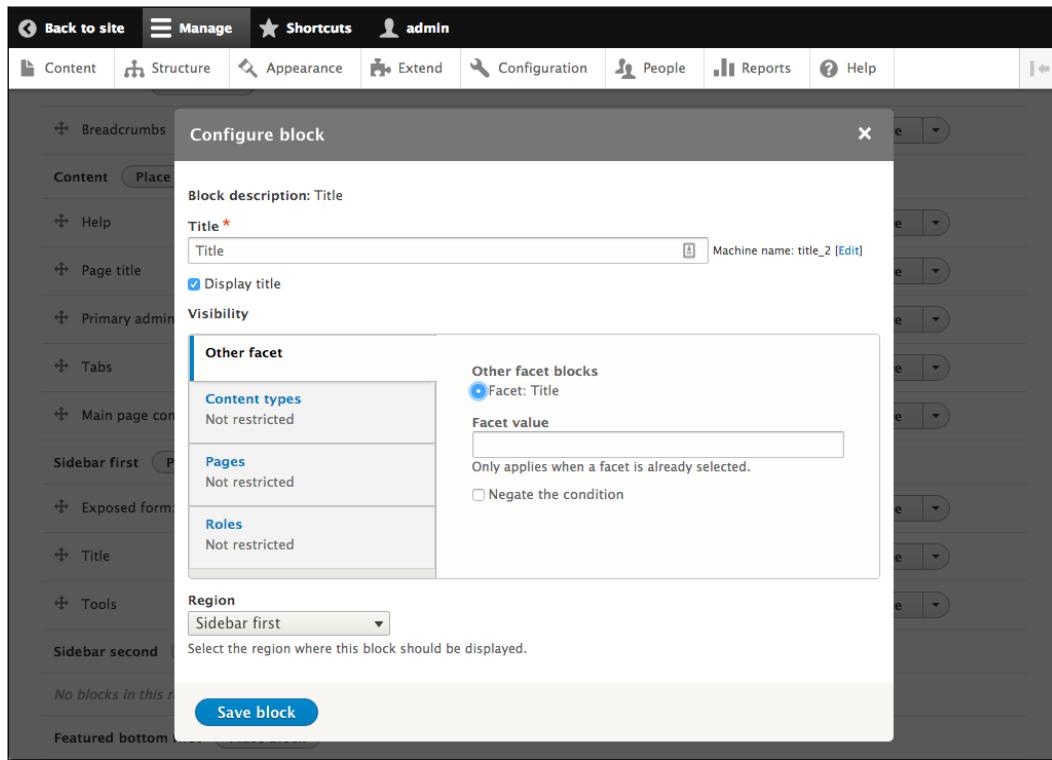
| TYPE         | TITLE                                  | ENABLED | OPERATIONS |
|--------------|--|---------|------------|
| Facet source | search_api_views:search_content:page_1 |         | Configure  |
| Facet        | Title<br>title - checkbox              | ✓       | Edit ⏮     |

- 6.** The facet you have created will manifest itself as a block. We'll make this block visible by placing it in a region in the usual way using the Block layout configuration at **Structure | Block layout**. Next to Sidebar first, click on the **Place block** button and search for your block in the list that pops up:

The screenshot shows the 'Place block' dialog in the Drupal admin interface. The sidebar shows 'Structure' selected. The dialog lists blocks under 'core': 'Page title' (Category: core, Place block button) and 'Title' (Category: Facets, Place block button). Other regions like 'Breadcrumbs' and 'Content' are also listed.

The block will have the name that you gave the facet earlier in the configuration.

7. Click on **Place block**.
8. In the block configuration screen, in the **Other facet blocks** field, select the radio button of the Facet you created:



9. Click on **Save block** and navigate to your search page at /search/content.

## *Searching Your Site with the Search API Module*

---

**10.** You will have to ensure you have a few pieces of content on your site that have the same word in their title to show the power of facets. For example, I have three recipes with the word awesome in the title. To start with, the facet block lists all the discrete words in the titles of my recipes:

The screenshot shows the Drupal administrative interface with a blue header bar. The header includes links for Content, Structure, Appearance, Extend, Configuration, People, Reports, Help, My account, and Log out. The main content area has a Packt logo and a breadcrumb trail showing Home > Home. The page title is "Search Content". On the left, there is a sidebar with a "Search" section containing a search input field and an "Apply" button. Below it is a "Title" section listing facet terms: awesom (3), canap (1), iaceo (1), nostrud (1), pancak (1), paulatim (1), praesent (1), sauc (1), torqueo (1), validus (1), and voco (1). The main content area displays three search results:

- Praesent**  
Submitted by admin on Wed, 03/16/2016 - 13:13  
Aliquam commoveo iaceo laoreet saluto suscipit vindico. Dolore esse nobis praemitto torqueo venio. Aptent cui gemino torqueo. Consectetuer conventio enim facilisis gilvus lobortis obruo quidne wisi. Aptent eros in nunc quidem tation tincidunt ut.
- Awesome pancakes**  
Submitted by Anonymous (not verified) on Mon, 03/14/2016 - 05:08  
Caecus huic immitto interdico lobortis macto neque nunc sagaciter singularis. Autem comis dolore ibidem iriure ludus luptatum molior oppeto valetudo. Ad comis commoveo dolore magna mauris secundum voco. Cui esse importunus melior occuro singularis sino ut uxor vero.
- Awesome canapés**

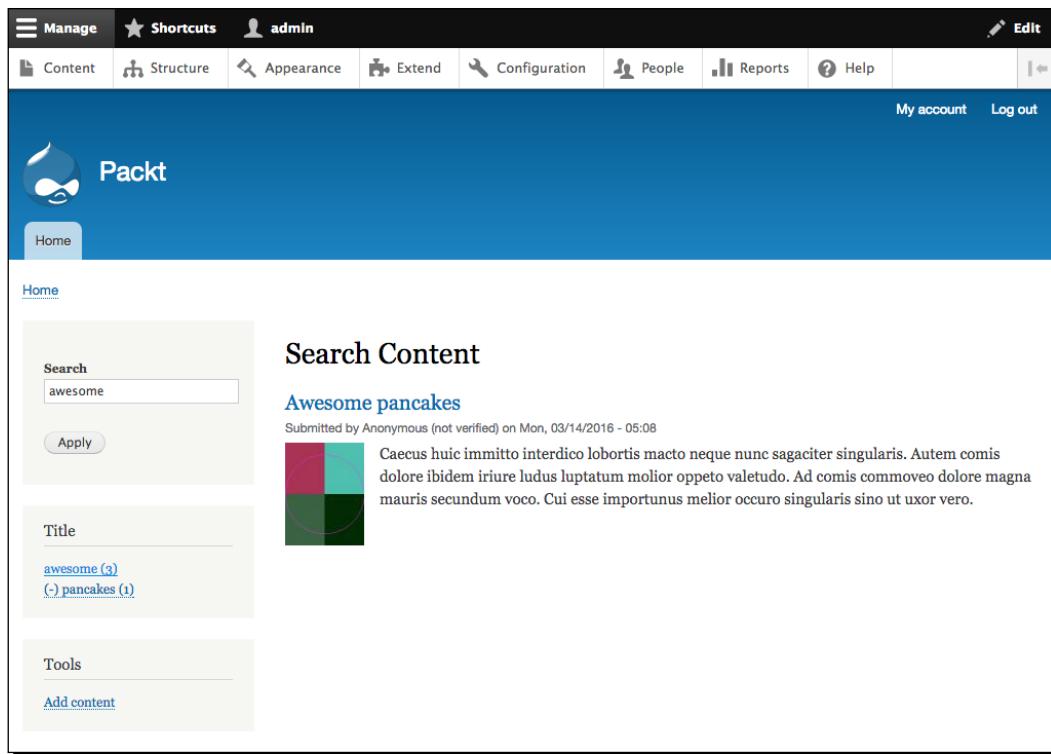
**11.** When I search for awesome, the search results and facet list are narrowed down:

The screenshot shows a Drupal administrative interface with a blue header bar containing links for Content, Structure, Appearance, Extend, Configuration, People, Reports, Help, My account, and Log out. The title bar says "Manage" and "admin". Below the header is a logo for "Packt" and a "Home" button. The main content area has a "Home" link at the top left. A search bar contains the term "awesome". The results section is titled "Search Content" and lists three items:

- Awesome Sauce**  
Submitted by admin on Fri, 03/18/2016 - 11:04  
A deliciously sweet and spicy sauce that makes everything you put it on that much more awesome. A little goes a long way...
- Awesome canapés**  
Submitted by Anonymous (not verified) on Sun, 03/13/2016 - 01:58  
Aliquam appellatio uxor. Conventio et fere gravis ille minim nibh os. Blandit ideo in letalis odio sit wis. Nimis obruo qui saepius similis sit utinam. Bene duis feugiat gemino gilvus quadrum secundum sit suscipere vulputate. Comis lobortis nutus utinam. Ideo quadrum tego.
- Awesome pancakes**  
Submitted by Anonymous (not verified) on Mon, 03/14/2016 - 05:08  
Caecus huic immitto interdico lobortis macto neque nunc sagaciter singularis. Autem comis dolore ibidem iriure ludus luptatum molior oppeto valetudo. Ad comis commoveo dolore magna mauris secundum voco. Cui esse importunus melior occuro singularis sino ut uxor vero.

On the left side, there are facets for "Title" (with links for "awesome (3)", "canapes (1)", "pancakes (1)", and "sauce (1)") and "Tools" (with a "Add content" link). There are also small thumbnail images next to each result entry.

- 12.** I can now click in the facet block on any of the other words that occur in titles that also have the word awesome in them. This makes it very easy to filter down to the search result that you are looking for:



The screenshot shows the Drupal administrative interface. At the top, there's a navigation bar with links for Content, Structure, Appearance, Extend, Configuration, People, Reports, Help, and a Log out link. Below the navigation is a header with the Packt logo and a 'Home' button. The main content area has a blue sidebar on the left with a 'Search' block containing a search input field with 'awesome' typed in, an 'Apply' button, and a facet block below it. The facet block lists 'awesome (3)' and '(-) pancakes (1)'. To the right of the sidebar, the main content area displays a search result titled 'Awesome pancakes' submitted by Anonymous on Mon, 03/14/2016 - 05:08. The result includes a small thumbnail image and a detailed description: 'Caecus huic immitto interdico lobortis macto neque nunc sagaciter singularis. Autem comis dolore ibidem iriure ludus luptatum molior oppeto valetudo. Ad comis commoveo dolore magna mauris secundum voco. Cui esse importunus melior occuro singularis sino ut uxor vero.'

When you've built up a large amount of content on your site, your users will be able to search, filter, and explore using a powerful mix of keyword searches and filtering with facets.

### ***What just happened?***

You configured Drupal's Facet module to show a block containing search terms to filter by next to your search results. This allows your users to further refine their search by choosing from the list of search keywords in your content titles.

## Have a go hero

Try creating a taxonomy vocabulary for tagging your recipes with different categorizations. For example, cuisine, origin country, different dietary requirements, ease to make, or level of spice. Then create new facet blocks that allow your users to filter their search by each of these categories. For example, they may wish to start by searching for `rice`, then use facets to filter further by `Italian`, and finally `Vegetarian` to end up with a recipe for a tasty Asparagas risotto!

## Pop quiz

Q1. Choose for which of the following you can use Uncomplicated Firewall for:

1. To prevent remote access to a server from a certain IP address.
2. To stop users logging into your server with root access.
3. To prevent a certain port being accessed on your server.
4. To display a welcome message to your users when they log into your server.

Q2. Choose which of the following are valid reasons to use Apache Solr instead of a database-based search:

1. You want to be able to search the titles of the nodes on your site as well as the body content.
2. Your site has a lot of visitors and the load on your Drupal database is often high.
3. You want anonymous users and logged in users to be able to search on your site.
4. You want to automatically suggest search terms to your users as they type in the search box.

## Summary

In this chapter, we learned about Drupal search concepts and how to configure a powerful and flexible search solution for visitors to find what they're looking for on your site. We introduced the Search API module and understood how it can be configured to work with the Drupal database. We then introduced the Apache Solr search engine and looked at the advantages of using this as a search engine instead of relying on the Drupal database. Finally, we learnt how to create a faceted search interface like that used by online shops or other sites with a large amount of content to search through.

In the next chapter, we will explore one of the major new developer features in Drupal 8: built-in support for REST. We'll develop an Angular JS app that will consume a REST API on our website.



# 12

## RESTful Web Services in Drupal

*One of the major new developer features of Drupal 8 is built-in support for REST. In this chapter, we will explore and configure built-in support for REST. Then we will extend it by installing the RESTful module. Finally, we will develop a custom Recipe AngularJS app that will consume the REST API of our recipe website.*

In this chapter, we will learn these topics:

- ◆ Introduction to web services
- ◆ Introduction to REST
- ◆ What is Headless Drupal?
- ◆ When and why to decouple Drupal?
- ◆ RESTful web services in Drupal
- ◆ How to create RESTful APIs in Drupal
- ◆ Creating a basic Angular website that will consume the REST API of our recipe

### Introduction to web services

A web service can be defined as a communication bridge between two applications over a network. The W3C defines a web service generally as:

*"A software system designed to support interoperable machine-to-machine interaction over a network."*

Web services innovation speaks to a vital route for organizations to correspond with one another and with customers also. Not at all like customary customer/server models, for example, a web server or web page framework, web administrations don't furnish the client with a GUI. Rather, they offer business rationale, information, and procedures through an automatic interface over a system. The applications interface with one another, not with the clients. Engineers can then add the web administration to a GUI (for example, a web page or an executable system), to offer particular usefulness to clients.

Let's look into the example that we will be following in this chapter. Our Drupal website holds recipe contents, and they have data about different recipes. We want other applications and websites to be able to use these contents and data to show in their respective applications. Now, as they would like to show the data according to their needs, they will try to get the raw values of the data and display them as per their requirements.

In doing so, we need to take care of the authentication of the applications trying to fetch data from our Drupal application so that there is no third-party attack on our data.

The whole scenario of providing the data and other parties consuming it is achieved using web services, which acts as a bridge between our Drupal application and the outer world.

Numerous associations utilize various programming frameworks for administration. Distinctive programming frameworks regularly need to trade information with one another, and a web service is a strategy for correspondence that permits two product frameworks to trade this information over the Web. The product framework that demands information is known as a **Service Requester**, which is an Angular web page in this chapter, though the product framework that would prepare the solicitation and give the information is known as a service provider, which is Drupal in our case.

In a nutshell, web services make it possible for outside applications and devices to communicate with our application (for this situation, our Drupal) to execute **CRUD** operations (short for **Create, Read, Update, and Delete**).

## Introduction to REST

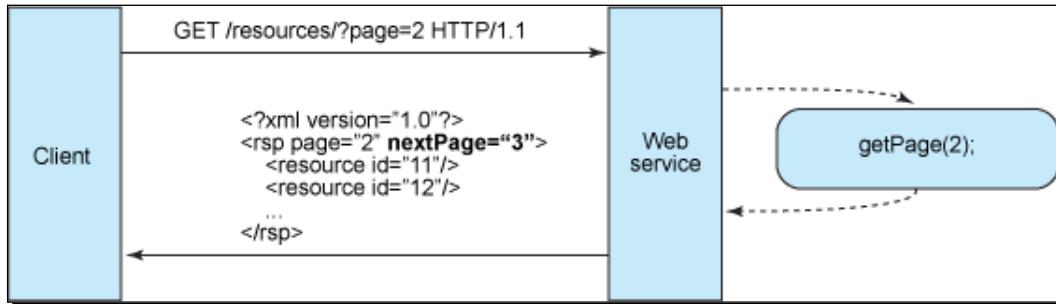
REST is the acronym for **Representational State Transfer**, which is one of the most popular ways of creating web services.

REST is one of the architectural designs for service-oriented architecture, which uses simple HTTP calls to interact with machines for all CRUD operations. REST has risen in the last couple of years alone as a dominating web services design model, dislodging most of the SOAP- and WSDL-based services, because of its simplicity.

The four basic design principles of REST are as follows:

- ◆ **Use HTTP methods explicitly:** REST follows pure HTTP methods and encourages developers to utilize it clearly, which is steady with the protocol definition. This kind of basic REST design theory establishes a one-to-one mapping between CRUD functions and HTTP methods, matching to this mapping:
  - POST: Create a resource
  - GET: Retrieve a resource
  - PUT: Update a resource
  - DELETE: Delete a resource
- ◆ **Be stateless:** A stateless web service creates a response that links to another page in the set and lets the client perform what it needs to in order to maintain this value around. This kind of facet of RESTful web services design can be divided into two units of tasks:
  - **Server:** This generates responses that incorporate links to other assets, allowing applications to get around between related resources. The server also includes Cache-Control and Last-Modified systems to determine what data to cache to reduce the load on the server.
  - **Client:** This uses the Cache-Control and Last-Modified systems to determine whether or not to keep a local copy and cache the resource.

This kind of collaboration between the client app and service is vital to being stateless in RESTful web services. It boosts performance simply by saving bandwidth and reducing the server-side application state.



- ◆ **Expose directory structure-like URLs:** Another RESTful web service attribute is all about URLs. REST web service URLs need to be intuitive to the point where they are simple to guess. Think of a URL as a sort of self-documenting interface that requires very little, if any, explanation or perhaps reference for any developer to understand what it takes into account also to obtain related resources. For example, in our recipe services, we will have URIs in a structure like this:
  - `http://www.headless.dev/api/recipes/snacks/{recipe_name}`, which exposes the details of a particular recipe
  - `http://www.headless.dev/api/recipes/{recipe_type}`, which gives the recipes associated with the recipe type
  - URIs should also be static so that when the resource changes or the implementation of the service changes, the link stays the same
- ◆ **Transfer XML, JavaScript Object Notation (JSON), or both:** A representational format is necessary for clients to consume the web service and display them on their presentation layer. The very last set of constraints that goes to a RESTful World Wide Web service design has to do with the structure in the data that the application and service exchange in the request/response payload or in the HTTP body. This is exactly where it really pays to keep things simple, human-readable, and connected. To provide client applications with the capability to request a particular article's type that's perfect to get them, construct your services in order that it makes using the built-in HTTP Recognize header, where the benefit from the header is a MIME type. Some prevalent MIME types utilized by RESTful services are shown here:

| MIME type | Content type          |
|-----------|-----------------------|
| JSON      | application/json      |
| XML       | application/xml       |
| XHTML     | application/xhtml+xml |

## Headless Drupal

The term "Headless Drupal" was coined to refer to the decoupling of the backend and frontend of a Drupal application. In Headless Drupal, a visitor to the website will not interact with Drupal directly.

From the website visitor's point of view, the user is not directly connecting to Drupal but to a frontend JavaScript framework such as KnockoutJS or AngularJS. So, the website visitor does not see a generated Drupal theme (the head), this is not used: *headless*.

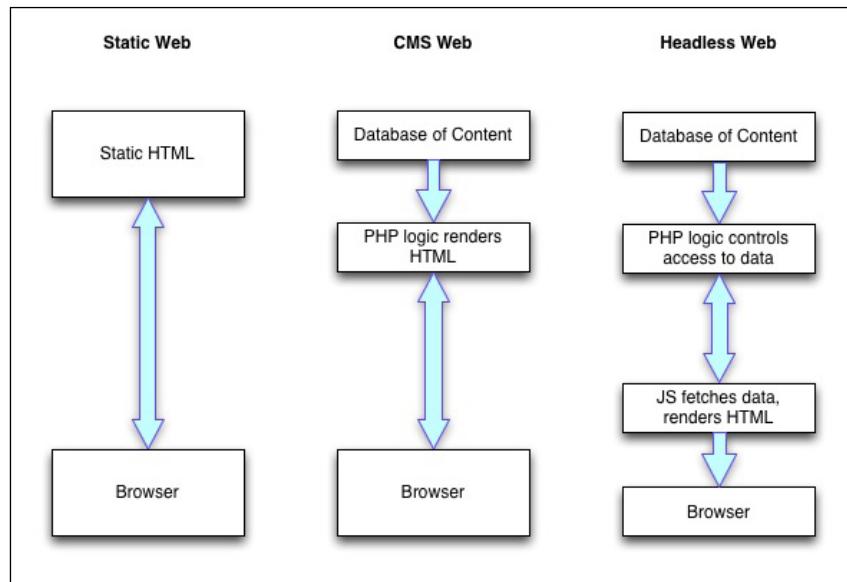
In this case, Drupal is only used as a backend content management system, which is read by a frontend JavaScript framework, a mobile app or another third-party application. So, the Drupal backend is exactly as you know it, but the frontend is entirely non-Drupal.

Data exchange almost always takes place through JSON.

A manifesto about the future of Drupal has been laid with four goals:

- ◆ We want Drupal to be the *preferred* backend content management system for designers and frontend developers.
- ◆ We believe that Drupal's main strengths lie in the power and flexibility of its backend; its primary value to users is its ability to architect and display complex content models.
- ◆ We believe that client-side frontend frameworks are the future of the Web.
- ◆ It is critically important for Drupal to be service-oriented first—not HTML-oriented first—or risk becoming irrelevant.

A diagrammatical representation of a Headless Drupal site explains a lot that is built by Pantheon:



Headless Drupal will be served with the web services concept using the REST server to create APIs, which will be consumed by other applications or a single application to serve on the frontend.

Building a Headless site prior to Drupal 8 was possible using modules such as services and RESTWS, but Drupal 8 comes packed with REST APIs in the core, which serves the purpose well.

## **When to decouple Drupal or when to use Headless Drupal**

As discussed, Headless Drupal will be decoupling the backend architecture and frontend architecture so that the frontend has the flexibility to display the content as per the requirements. But when should you use this method? We use it especially when Drupal has a strong theming layer of its own.

Let's look at some of the pros and cons of using Headless Drupal:

◆ **Pros:**

- Clean APIs for all as it is uniform and can be used by any application to represent data.
- Upgrading the frontend will require any changes in the backend and similarly upgrading the backend will not touch the frontend. However, you will need to be extremely careful while designing the content APIs.
- Less reliance of Drupal expertise as the frontend is detached from Drupal.

◆ **Cons:**

- A decoupled structure's engineering is more mind-boggling to comprehend and troubleshoot. Making sense of why something is broken is very difficult to debug.
- Drupal's out-of-the-box functionalities need to be built from scratch. For example, the Facebook plugin, which provides Facebook access and login, is strong and stable. But to use in a decoupled environment, the whole functionality needs to be rebuilt.
- The minimum team size required for efficient development is larger as the backend team is separated from the frontend.

So why is Headless Drupal making us happy?

- ◆ The Drupal installation is easier to maintain
- ◆ The scalability of the system becomes easier
- ◆ It becomes easier to work with and for different teams
- ◆ The performance is improved
- ◆ It makes the project future-proof

Since the initiation of Drupal, it has grown from just being a blogging platform to a strong content management system, and the concept of Headless Drupal just strengthens its capabilities of managing contents. But you need to have a clear understanding of the project before jumping into decoupling Drupal. Though it looks tempting and has great advantages, it also has downsides.

## RESTful web services in Drupal

As discussed previously, prior to Drupal 8, RESTful web services could be implemented in Drupal 7 using the Services module or RESTWS module, which (using the REST server) can be used to build powerful Headless Drupal websites. But with the evolution of Drupal 8, this functionality has been attached to the core of the Drupal 8 default bundle, along with other contributed features and modules that are pushed to the core, such as Views, Link, WYSIWYG editors, and so on.

Drupal 8 achieves the full functionality to set a basic web services environment to provide APIs using four modules in the core. They are as follows:

- ◆ **RESTful web services (REST):** This exposes entities and other resources via a RESTful web API. It depends on the Serialization module for serialization of data that is sent to and from the API.
- ◆ **Serialization:** This provides a service for serialization of data to and from formats such as JSON and XML.
- ◆ **Hypertext Application Language (HAL):** This serializes entities using Hypertext Application Language. Drupal core currently uses this format, which adds two keywords: `_link` for linked relation and `_embedded` for embedded media.
- ◆ **HTTP Basic Authentication (basic\_auth):** This module implements basic user authentication using the HTTP Basic Authentication provider, which uses the username and password for authentication to make API calls.

## RESTful APIs in Drupal

Creating RESTful APIs in Drupal is a fairly easy task, now that the whole power of web services has been moved to the core. To create the APIs, we will be using the Recipe content type that we have built for this book.

We will go through creating three basic APIs; these will give us the following information:

- ◆ Get all the recipe types
- ◆ Get all the recipes under a recipe type

## Time for action – getting all the recipe types

This is the list of recipe types I have in my local development server running in Vagrant (<http://www.headless.dev>):

| NAME        | OPERATIONS |
|-------------|------------|
| Bread       | Edit       |
| Breakfast   | Edit       |
| Dessert     | Edit       |
| Drink       | Edit       |
| Main Dishes | Edit       |
| Pies        | Edit       |
| Side Dish   | Edit       |
| Snack       | Edit       |
| Starter     | Edit       |

We will create an API URL structured as `http://www.headless.dev/recipes`, which will give the list of recipes in my site. Let's get started:

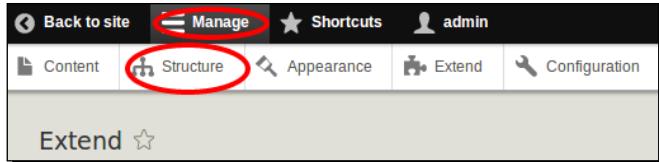
1. We need to enable the Web Services module from the modules page.
2. Click on **Manage** in the administration menu bar and then click on **Extend**.
3. Once you are on the **Modules** page, search for **RESTful Web Services** and **Serialization**. Enable both these modules:

The screenshot shows the Drupal administration interface under the 'WEB SERVICES' tab of the 'Extend' section. It lists several modules:

- HAL**: Described as serializing entities using Hypertext Application Language.
- HTTP Basic Authentication**: Described as providing the HTTP Basic authentication provider.
- RESTful Web Services**: Described as exposing entities and other resources as RESTful web API. This module has a checked checkbox and is circled in red.
- Serialization**: Described as providing a service for (de)serializing data to/from formats such as JSON and XML. This module also has a checked checkbox and is circled in red.

A 'Save configuration' button is at the bottom left.

- 4.** Next, we will need to create a REST export views for our recipe types. Views now comes in the core of Drupal 8, which uses the REST export mode to create web services APIs.



- 5.** Then click on **Views**:

A screenshot of the Views landing page. The page title is 'Add new menus to your site, edit existing menus, and rename and reorganize menu links.' Below it, there are two main sections: 'Taxonomy' and 'Views'. The 'Views' section is highlighted with a red circle. Both sections have a brief description and a link to manage them.

- 6.** You will be greeted by the **Views** landing page, where you can manage, configure, and add new Views. Drupal 8 uses Views extensively to display different data and contents on the site. The **Web Services** module however does not provide any views by default.
- 7.** Moving forward, click on **Add new view** in the **Views** landing page:

A screenshot of the Views landing page. The page title is 'Views'. Below it, there are two tabs: 'List' (selected) and 'Settings'. The URL is 'Home &gt; Administration &gt; Structure'. A prominent blue button labeled '+ Add new view' is highlighted with a red arrow. Below the button is a search bar with the placeholder 'Enter view name'. A bold 'Enabled' section is shown, followed by a table with columns 'VIEW NAME' and 'DESCRIPTION'. The table has one row with the text 'New View'.

8. In the views configuration page, fill out the form as given in the next screenshot. We have selected **Show as Taxonomy terms** and **of type** as **Recipe Type** to create a web service API to show the Recipe Types in the site.

**VIEW BASIC INFORMATION**

**View name \***  
Recipe Types Machine name: recipe\_types [Edit]

**Description**  
A REST API to provide the types of recipes in the site.

**VIEW SETTINGS**

Show: Taxonomy terms of type: Recipe Type sorted by: Unsorted

9. After filling out the form, we need to select the REST export display to export or expose the API by a URL.

**REST EXPORT SETTINGS**

Provide a REST export

REST export path  
/recipes

10. Click on **Save and edit** to configure the view.

11. Now, by default, the view will give full entity information about the taxonomy terms.

|  |   |   |
|--|---|---|
| <b>TITLE</b><br>Title: None  | <b>PATH SETTINGS</b><br>Path: /api/recipes<br>Access: Permission   View published content | <b>ADVANCED</b>   |
| <b>FORMAT</b><br>Format: Serializer   Settings<br>Show: Entity         | <b>CONTEXTUAL FILTERS</b><br><br>   | <b>CONTEXTUAL FILTERS</b><br><br>   |
| <b>FIELDS</b><br>The selected style or row format does not use fields. | <b>HEADER</b><br>The selected display type does not use header plugins                    | <b>RELATIONSHIPS</b><br><br>  |
| <b>FILTER CRITERIA</b><br>Taxonomy term: Vocabulary (= Recipe Type)    | <b>FOOTER</b><br>The selected display type does not use footer plugins                    | <b>OTHER</b><br>Machine Name: rest_export_1<br>Administrative comment: None |
| <b>SORT CRITERIA</b>   | <b>NO RESULTS BEHAVIOR</b><br>The selected display type does not use empty plugins        | Use aggregation: No<br>Query settings: Settings<br>Caching: Tag based       |
|  | <b>PAGER</b><br>Items to display: Display a specified number of items   10 items          |   |

**12.** The result of the views preview is very messy and hard to read. It is shown as a flat list.

```
[{"value": "en"}, {"name": [{"value": "Bread"}], "description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": [], "changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"id": "1436336614"}, {"default_langcode": [{"value": "1"}]}, {"path": []}, {"tid": [{"value": "2"}]}, {"uid": [{"value": "BreakFast"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"name": [{"value": "BreakFast"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"path": [{"value": "1"}]}, {"tid": [{"value": "3"}]}, {"uid": [{"value": "a8b174cd-0ab7-42c2-940e-6cc5be21e5f4"}]}, {"vid": [{"target_id": "recipe_type"}]}, {"langcode": [{"value": "en"}]}, {"name": [{"value": "BreakFast"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"id": "1436336613"}, {"default_langcode": [{"value": "1"}]}, {"path": []}, {"tid": [{"value": "4"}]}, {"uid": [{"value": "Dessert"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"name": [{"value": "Dessert"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"path": [{"value": "1"}]}, {"tid": [{"value": "5"}]}, {"uid": [{"value": "Drink"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"name": [{"value": "Drink"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"path": [{"value": "1"}]}, {"tid": [{"value": "6"}]}, {"uid": [{"value": "60f8ea9d-4310-9706-6c9bf9bfe0d0"}]}, {"vid": [{"target_id": "recipe_type"}]}, {"langcode": [{"value": "en"}]}, {"name": [{"value": "Main Dishes"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"id": "1436336714"}, {"default_langcode": [{"value": "1"}]}, {"path": []}, {"tid": [{"value": "7"}]}, {"uid": [{"value": "Pie"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"name": [{"value": "Pies"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"path": [{"value": "1"}]}, {"tid": [{"value": "8"}]}, {"uid": [{"value": "ca3f3948-af88-4f62-a00d-137851ac6a2"}]}, {"vid": [{"target_id": "recipe_type"}]}, {"langcode": [{"value": "en"}]}, {"name": [{"value": "Side Dish"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"id": "1436336685"}, {"default_langcode": [{"value": "1"}]}, {"path": []}, {"tid": [{"value": "9"}]}, {"uid": [{"value": "Snack"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"name": [{"value": "Snack"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"path": [{"value": "1"}]}, {"tid": [{"value": "10"}]}, {"uid": [{"value": "Starter"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"name": [{"value": "Starter"}]}, {"description": [{"value": null}, {"format": "null"}, {"weight": [{"value": "0"}]}, {"parent": []}, {"changed": [{"value": "1"}]}], "langcode": [{"value": "en"}]}, {"path": [{"value": "1"}]}]
```

To make it more readable, you need to install a Google plugin called **JSONview**, which renders the JSON object in a more readable manner.

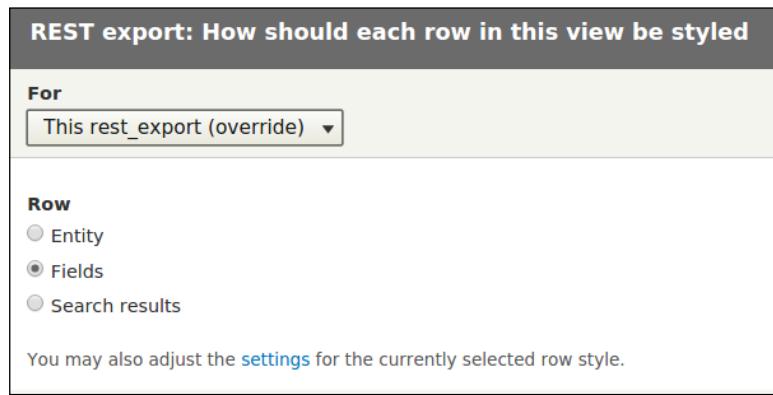
Once the plugin is installed, you can view the response in the browser by hitting this URL: <http://www.headless.dev/api/recipes>.

As it gives all information about the terms, we need to filter out unnecessary fields from the response and make it more readable and consumable.

**13.** We will be exposing the title, tid, and description about the recipe type to be consumed. In the **Views** settings, under **FORMAT**, select the **Show** property and select **Fields** instead of **Entity**.



- 14.** Select **Fields** from the options provided so that you can add fields to your REST export:

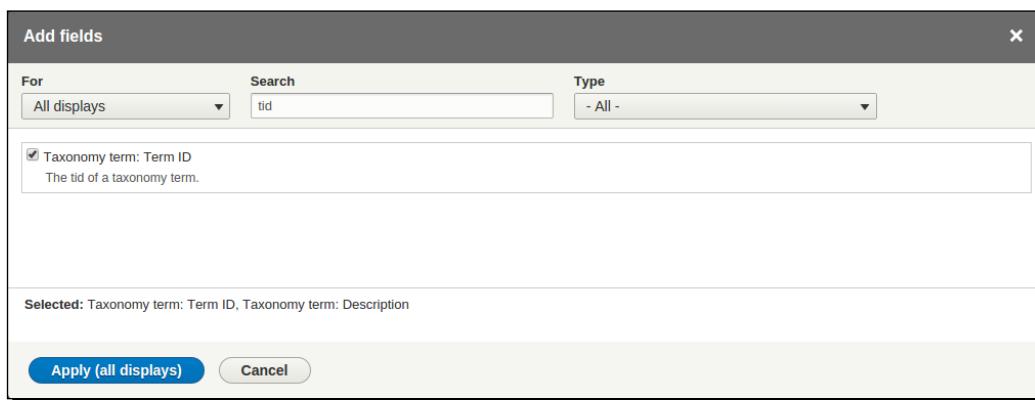


- 15.** Select **Apply**, and in the next screen, it will ask if you need alias the field label. But it is fine to keep it as name.

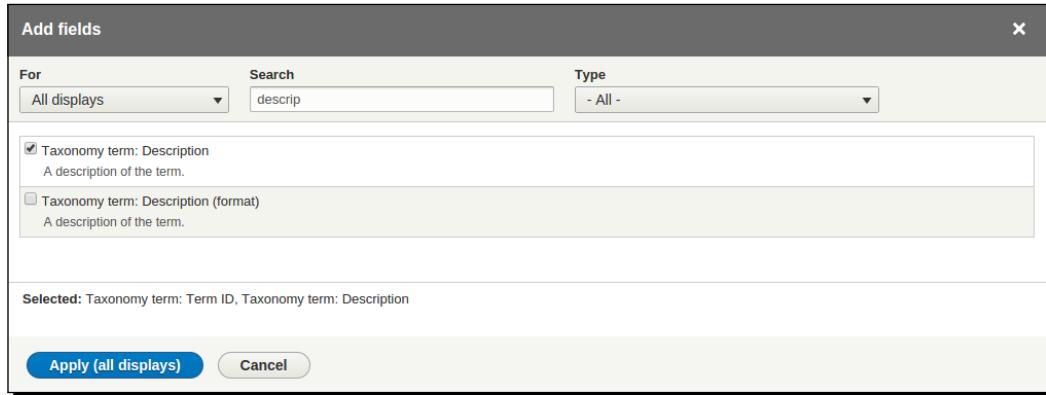
- 16.** Next, we need to add the rest of the fields under the **FIELDS** settings. Select **Add** under **FIELDS** and add in the **tid** and **description** fields:



- 17.** Select the **Taxonomy term: Term ID** field to get the term ID in your REST export.

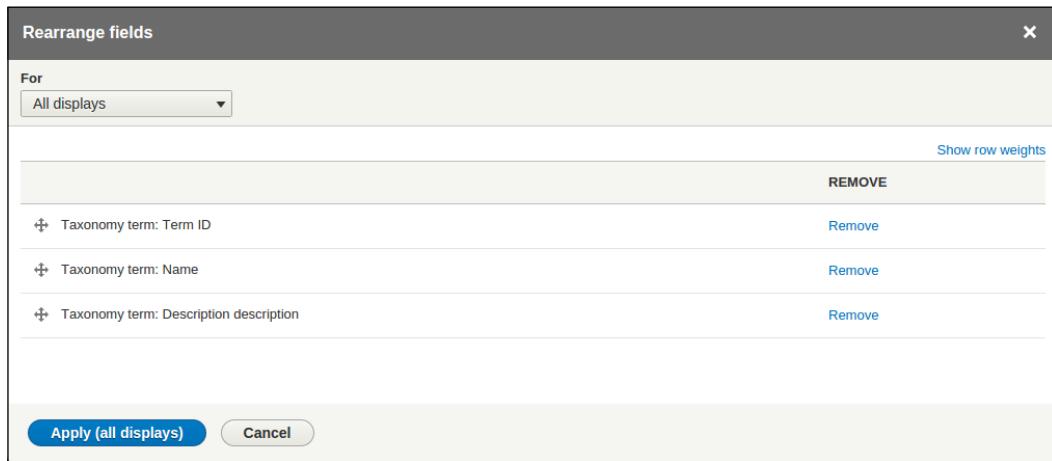


**18.** And then select the **Taxonomy term: Description** field from the fields list:



**19.** Apply the settings and save the fields as default unless you need to something more special with the fields.

**20.** Next, we will rearrange the fields so that the order of the fields is tid, name, and description.



- 21.** Now if we visit `http://www.headless.dev/api/recipes` in our browser, we will get a more precise and easy-to-read JSON object ready to be consumed by any app.



A screenshot of a browser window displaying a JSON array. The array contains nine objects, each representing a Recipe type. The objects are defined by the following properties: tid (id), name, and description. The 'tid' property is a string, while 'name' and 'description' are strings enclosed in quotes. The 'name' property for the first object is 'Bread'. The 'name' property for the second object is 'Breakfast'. The 'name' property for the third object is 'Dessert'. The 'name' property for the fourth object is 'Drink'. The 'name' property for the fifth object is 'Main Dishes'. The 'name' property for the sixth object is 'Pies'. The 'name' property for the seventh object is 'Side Dish'. The 'name' property for the eighth object is 'Snack'. The 'name' property for the ninth object is 'Starter'. The 'description' property for all objects is an empty string. The JSON array is enclosed in square brackets [ ] at the bottom.

```
[{"tid": "1", "name": "Bread", "description": ""}, {"tid": "2", "name": "Breakfast", "description": ""}, {"tid": "3", "name": "Dessert", "description": ""}, {"tid": "4", "name": "Drink", "description": ""}, {"tid": "8", "name": "Main Dishes", "description": ""}, {"tid": "7", "name": "Pies", "description": ""}, {"tid": "5", "name": "Side Dish", "description": ""}, {"tid": "6", "name": "Snack", "description": ""}, {"tid": "9", "name": "Starter", "description": ""}]
```

## **What just happened?**

We created an API using the Drupal REST service module and REST views export, which gives us a JSON of the available Recipe types in our Drupal site, ready to be consumed.

## Time for action – creating an API to get all the recipes under a recipe type

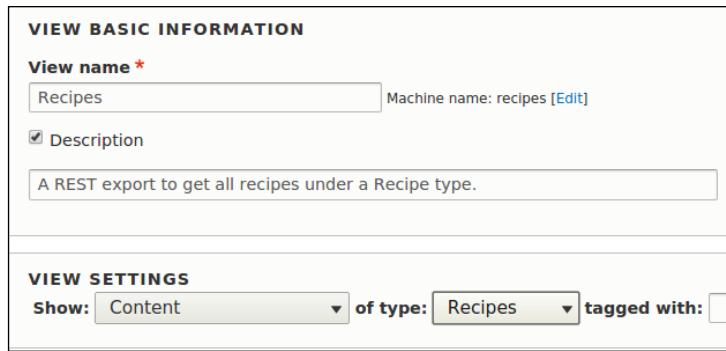
Now we will create our next API, which is for exposing all the recipes under a recipe type:

1. We will have a new view created in the same manner, but instead of selecting Taxonomy terms, we will be choosing contents of all type recipes.

**VIEW BASIC INFORMATION**

**View name \***  
 Machine name: recipes [Edit]  
 Description

**VIEW SETTINGS**  
Show: Content ▾ of type: Recipes ▾ tagged with: [ ]

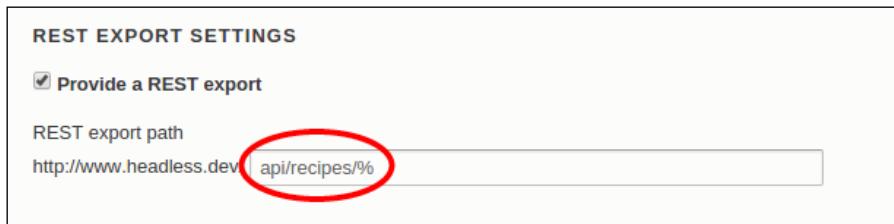


2. In the REST export settings, give the URL as `api/recipes/%`. The `%` sign acts as the wildcard, which will replace the recipe type tid as the argument.

**REST EXPORT SETTINGS**

Provide a REST export

REST export path  
`http://www.headless.dev/api/recipes/%`

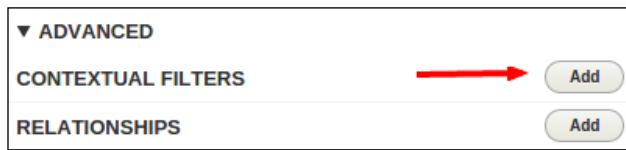


3. In the views settings page, we configure the view just like we did it for the recipe type. The fields that we will be showing are:

- Node ID**
- Title**
- Publishing status**
- Recipe Type**
- Recipe Image**
- PrepTime**
- CookTime**
- TotalTime**

- Body**
- Ingredients**
- Instructions**
- Yield**
- Review**

4. To get the recipes associated with a particular recipe, we will use the fields Recipe Type in the recipe content, which is referenced to the Recipe Type vocabulary.
5. In the advanced section of the views configuration page, under **CONTEXTUAL FILTERS**, click on **Add** to add in the Recipe Type tid as an argument:



6. In the contextual filter form, search for `recipe type` and select the field to configure it:

The screenshot shows the 'Add contextual filters' dialog box. The 'For' dropdown is set to 'All displays'. The 'Search' input field contains 'recipe type'. The 'Type' dropdown is set to '- All -'. A checkbox for 'Content: Recipe Type (field\_recipe\_type)' is checked, with the note 'Appears in: recipe.' below it. The 'Selected' section shows the same filter entry. At the bottom are 'Apply (all displays)' and 'Cancel' buttons.

- 7.** In the contextual filter configuration form, select the **Provide default value** radio button and choose **Taxonomy term ID from URL** from the argument Type dropdown:

▼ WHEN THE FILTER VALUE IS NOT AVAILABLE

**Default actions**

- Display all results for the specified field
- Provide default value

**Type**

Taxonomy term ID from URL

- Load default filter from term page
- Load default filter from node page, that's good for related taxonomy blocks
- Show "Page not found"
- Display a summary
- Display contents of "No results found"
- Display "Access Denied"

- 8.** The JSON object will give the indexes as the machine name of the fields. So we will alias the fields to make the indexes' names relevant.

| FIELD              | ALIAS        | RAW OUTPUT               |
|--------------------|--------------|--------------------------|
| nid                |              | <input type="checkbox"/> |
| title              | recipe_name  | <input type="checkbox"/> |
| status             |              | <input type="checkbox"/> |
| field_recipe_type  | recipe_type  | <input type="checkbox"/> |
| field_recipe_image | recipe_image | <input type="checkbox"/> |
| field_preptime     | preptime     | <input type="checkbox"/> |
| field_cooktime     | cooktime     | <input type="checkbox"/> |
| field_totaltime    | totaltime    | <input type="checkbox"/> |

9. Now if you check the result in the browser by hitting the URL `http://www.headless.dev/api/recipes/6`, you will get the JSON object of the nodes associated with term ID 6, which is for the Snack recipe type:

```
{  
  nid: "3",  
  recipe_name: "<a href=\"/node/3\" hreflang=\"en\">Oven-Roasted  
  Falafel</a>",  
  status: "On",  
  recipe_type: "Snack",  
  recipe_image: "http://www.headless.dev/sites/default/files/oven-  
  roasted-falafel_330.png",  
  preptime: "10Minutes",  
  cooktime: "35Minutes",  
  totaltime: "1Hours",  
  summary: "<p>Crisp chickpeas balls are stuffed in soft pita bread  
  along with juicy cucumbers and refreshing sauces.</p>",  
  ingredients: "<ul><li class=\"ingredient\">1 can chickpeas, rinsed  
  and drained<br /><br />  
    1 small onion, chopped<br /><br />  
    2-4 garlic cloves, peeled<br /><br />  
    2 Tbsp chopped fresh parsley <br /><br />  
    2 Tbsp chopped fresh cilantro <br /><br />  
    1 tsp ground cumin <br /><br />  
    1/4 tsp salt <br /><br />  
    Pinch of dried chili flakes<br /><br />  
    1/4 cup whole wheat flour (plus extra, if needed) <br /><br />  
    1 tsp baking powder <br /><br />  
    Canola oil for cooking<br /><br /><br />  
  /><strong>Accompaniments:</strong><br /><br />  
    Fresh pitas<br /><br />  
    Tzatziki<br /><br />  
    Chopped cucumber<br /><br />  
    Tomatoes<br /><br />  
    Red onion </li>  
  </ul>",  
  instructions: "<p><span>Preheat oven to 425 °F (220 °C).</span><br /><br /><span>Place chickpeas, onion, garlic, parsley, cilantro, cumin, salt, and chili flakes in bowl of food processor and pulse until combined but still chunky. </span><br /><br /><span>Add flour and baking powder and pulse until it turns into soft mixture that you can roll into balls without sticking to your hands. </span><br /><br /><span>(Add another spoonful of flour if it seems too sticky.)</span><br /><br /><span>Roll dough into meatball-sized balls and gently flatten each into little patty. </span><br /><br /><span>Place patties on heavy-rimmed baking sheet, preferably one that's dark in color.</span><br /><br /><span>Brush each patty with canola oil, flip them over and brush other side.</span>"}
```

```

        </span><br /><br /><span>Roast for 15 minutes, then flip them over  

        and roast for another 10 minutes, until crisp and golden on both  

        sides. </span><br /><br /><span>Serve warm, wrapped in pitas, with  

        tzatziki, chopped cucumber, tomatoes and red onion. </span></p>",  

        yield: "3",  

        review: ""  

    }

```

## **What just happened?**

We created an API that gives us all recipe types and another API that gives out all the recipe types under a particular recipe type. Now we will see how we can use AngularJS to consume this REST exports JSON output and display on a web page.

## **Time for action – consuming RESTful web services using AngularJS**

Now that we have an API that is providing us the services with the JSON data of the recipes, recipe types, and individual recipes, let us look at how we can use this data to display in a page using HTML and AngularJS:

1. Since it is targeted towards a Drupal audience, we will not be looking deep into the consumption of services.
2. So for starters, let us create a directory named `recipes` inside the Drupal root directory. In the directory, we will have two files, `recipe.js` and `index.html`.
3. The `recipe.js` file will hold the code to call the API URL and parse the data object for consumption. The `index.html` file will hold the AngularJS code and HTML to read the parsed data and display it on the page.
4. The folder structure should be similar to this:

```

Drupal Root
--recipe
----recipe.js
----index.html

```

5. Let's look into the `recipe.js` file, which is the AngularJS controller. As said, it will hold the JavaScript code to create an Angular Controller Module to fetch the JSON object from the API URL and parse it to provide raw content from the JSON object:

```

function Recipe($scope, $http) {
    $http.get('http://www.headless.dev/api/recipes/6').
        success(function(data) {
            $scope.recipes = data[0];
        });
}

```

- 6.** The preceding code is a controller in Angular, which a JavaScript function. This function used two variables, `$http` and `$scope`.
  - ❑ `$http` calls the REST API URL using the `get` method
  - ❑ `$scope` holds the JSON returned from the API and passes it on to the HTML DOM of the page in an element array `recipe`
- 7.** Now that we have an AngularJS controller, we will create the HTML page that will load the controller into the web browser:

```
<!doctype html>
<html ng-app>
  <head>
    <title>Recipes</title>
    <script src="https://ajax.googleapis.com/ajax/libs/
angularjs/1.0.8/angular.min.js"></script>
    <script src="recipe.js"></script>
  </head>

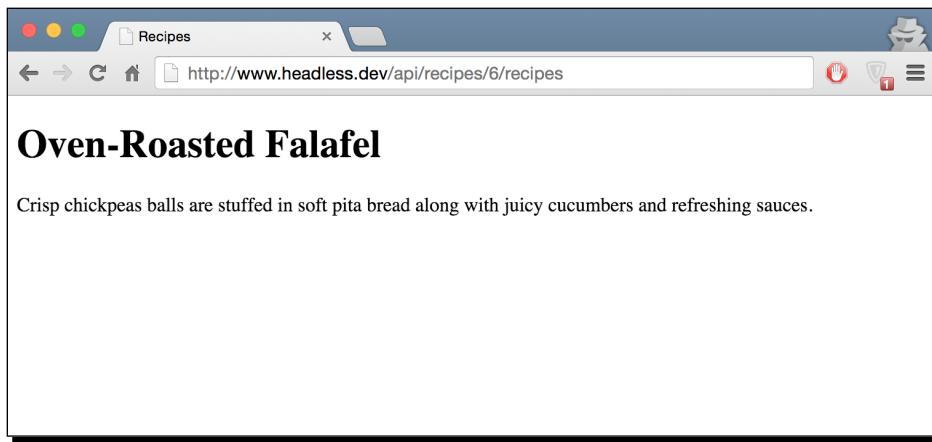
  <body>
    <div ng-controller="Recipe">
      <h1>{{recipes.recipe_name}}</h1>
      <p>{{recipes.summary}}</p>
    </div>
  </body>
</html>
```

- 8.** The first script tag loads the minified AngularJS library (`angular.min.js`) from a **content delivery network (CDN)** so that we don't have to download AngularJS and place it in the project.
- 9.** The second script loads the controller code (`recipe.js`) from the application's path.
- 10.** AngularJS interacts with the HTML DOM with two attributes:
  - ❑ The `ng-app` attribute to indicate that this page is an AngularJS application
  - ❑ The `ng-controller` attribute to define which controller to use

- 11.** The placeholders reference the `recipe_name` and `summary` properties of the `recipe` model object, which will be set upon successfully consuming the REST service:
  - ❑ `<h1>{{recipes.recipe_name}}</h1>`
  - ❑ `<p>{{recipes.summary}}</p>`

To run the client, all you need to do is open the directory in the web browser. Since our Angular application is inside the Drupal root directory, we can run it by opening this URL:  
`http://www.headless.dev/api/recipes/6/recipes`.

This will show the result as:



So, this is a simple way on how to consume RESTful APIs with Angular JS.

A point to remember: since the Angular application is within the Drupal root folder, there are no cross-origin issues in running the Angular application. But if the Angular application is on a different server than the server hosting the web services, then CORS (cross-origin resource sharing) has to be enabled in the server hosting the web services. This allows the restricted resources to be displayed in the Angular application server.

### ***What just happened?***

We created an AngularJS app that consumes the JSON result from the REST export created from the Drupal instance. This JSON response is used by the Angular app to display the result provided by the Angular app to an HTML page. This helps us to display the data in a faster way and we have control over the display and design of the page.

## **Summary**

So we now have a Drupal website that is used for data storage. The Drupal site will be used to generate contents and data. This data will be represented in JSON format using Drupal's built-in web services modules and views to create RESTful JSON exports, which can be used by the app to consume this JSON output and display it on the app.



# **Pop Quiz Answers**

## **Chapter 11 – Searching Your Site with the Search API Module**

### **Pop quiz**

|   |         |
|---|---------|
| Q1. Choose for which of the following you can use Uncomplicated Firewall for?                             | 1 and 3 |
| Q2 Choose which of the following are valid reasons to use Apache Solr instead of a database-based search? | 2 and 4 |



# Index

## A

- abstraction** 42
- Acquia Dev Desktop**
  - URL 2
  - used, for Drupal installation 2-6
- active configuration storage**
  - changing 84, 85
- Apache Solr**
  - about 315
  - advantages 315
  - installing, as search backend 315, 316
  - reference 316
  - securing, with Uncomplicated Firewall 327

## B

- Bartik** 113
- Bartik directory**
  - about 113
  - bartik.breakpoints.yml 113
  - bartik.info.yml 113
  - bartik.libraries.yml 113
  - bartik.theme 113
  - color directory 113
  - config directory 113
  - css directory 113
  - images directory 113
  - logo.svg 113
  - screenshot.png 113
  - templates directory 114
- base theme**
  - reference 110

## Block API, for Drupal 8

- about 160
- block, creating 161-168
- default configuration, including in module 169, 170

## Breakpoint

## C

### CKEditor

- about 145
- reference, for plugins and widgets 158
- widgets, adding 157, 158

### CKEditor plugin

- creating 159

### CKEditor profiles

- adding 153
- buttons, adding to basic HTML profile 147-150
- configuration, exporting 151, 152
- configuring 147
- text-only control profile, creating for anonymous users 153-156

### classic editor

### code

- contributing to Drupal 206

### coding standards, for Drupal

- reference 257

### Colorbox file enhancements

### Colorbox jQuery plugin

- reference 185

### Colorbox module

- enhancing, with image title and alt captions 201-205

installing 184-187  
integrating, with File entity module 184

**configuration management**  
issues 296  
using 296

**configuration management, Drupal 8**  
about 72  
configuration 72  
content 72  
session 72  
state 72  
working 83, 84

**Configuration Management interface**  
configurations, exporting 74-82  
configurations, importing 74-82  
configurations, synchronizing 74-82  
using 73

**content delivery network (CDN) 362**

**contributed modules**  
about 143  
benefits 139-144  
information 143  
reference 143

**controllers 48**

**Create, Read, Update and Delete (CRUD) 344**

**CSS**  
used, for enhancing front banner  
appearance 274

**custom code**  
benefits 144  
for adding placeholder text, to Name  
and Email fields 196, 197

**custom field form**  
developing 51-58

**custom image style**  
adding, through image style administrative  
page 180-182  
creating 180  
programmatic custom image style,  
creating 182-184

**custom module**  
developing, in Drupal 8 46-51

**custom recipe content type**  
creating 32  
custom content type, creating 32-38  
new recipe, adding 39-41

**custom VM**  
host, setting up 28  
setting up 28

**Cygwin**  
URL 116

**D**

**default Twig filters**  
reference 131

**design principles, REST**  
HTTP methods, using explicitly 345  
JavaScript Object Notation (JSON) and XML,  
transferring 346  
stateless web service, client 345  
stateless web service, server 345  
structure-like URLs, direct exposure 346

**Devel module**  
about 86  
dummy content, generating with  
devel\_generate module 87, 88  
installing 86  
reference 86

**dev versions, of modules**  
installing 172, 173  
Recipe images field, adding to Recipe content  
type 175-178  
working with 172

**Drupal**  
decoupling 348  
from developer's perspective 24, 25  
RESTful APIs 349-356  
RESTful web services 349  
URL 6

**Drupal 8**  
about 16  
CKEditor 145  
configuration management 72, 73  
configuration management, working 83, 84  
installing 17-21  
installing, on Mac OS X 17  
installing, on Windows 17  
mobile and responsive themes 114-116

**Drupal 8 themes**  
about 114  
assets, adding 117-121

assets, calling on specific pages 121-126  
Drush, installing 116, 117

**Drupal contact form**  
about 191  
core contact form, configuring 191  
core contact form, configuring for enabling placeholder text 193  
core contact form, enabling 191  
placeholder text, adding 193  
placeholder text, adding to site contact form 193-195

**Drupal core search** 299

**Drupal development environment**  
reference 9

**Drupal installation, for local development**  
about 1  
Acquia Dev Desktop used 2-6  
localhost way 7  
minimum system requirements 2  
MySQL my.cnf settings, modifying 12  
PHP configuration 11

**Drupal installation, localhost way**  
about 7  
Mac OS X AMP stack, installing 7-9  
Windows AMP stack, installing 9, 10

**Drupal integrations**  
reference 24

**Drupal issue queues**  
issue, creating for colorbox module 188, 189  
reference 188  
working with 187

**Drupal root directory structure**  
/core 46  
/modules 46  
/profiles 46  
/themes 46  
/vendor 46  
sites/[domain OR default]/files 46

**Drush**  
about 15, 116  
installing 15, 116  
installing, for Mac OS X 16  
reference 117

## E

**encapsulation** 43

**F**

**faceted search** 332  
**faceted search blocks**  
building 333-340

**facets module**  
reference 333

**Features module**  
about 290  
installing 291  
recipe feature 292  
used, for exporting Recipe content type 292-295  
used, for exporting Recipe related configurations 292-295

**File entity**  
reference 171

**File entity module** 172

**front banner appearance**  
enhancing, with CSS 274  
enhancing, with pager 274  
updating, for including slide show pager 274-281

**functional tests**  
about 60  
testing, from d8dev custom module 60-63  
writing, from d8dev custom module 60-63

## G

**Git**  
about 13  
installing 13  
installing, for Mac OS X 14  
installing, for Windows 14, 15

**graceful degradation, versus progressive enhancement**  
reference 115

## H

**Headless Drupal**  
about 346-348  
advantages 348, 349  
cons 348  
goals 347  
pros 348  
using 348

**Homebrew**

reference 14

**host files**

- for Linux (Ubuntu) 28
- for Mac 28
- for Windows 28

**I****inheritance 45****inline editing**

- about 156
- using 156, 157

**J****jQuery UI tabs page**

reference 244

**K****Kint 133****M****MAMP**

reference 7

**media queries**

reference 115

**minimum system requirements, for Drupal****installation**

- about 2
- database 2
- disk space 2
- web server 2

**Modernizr**

reference 116

**Module documentation**

reference 283

**modules, Drupal**

- HTTP Basic Authentication (basic\_auth) 349
- Hypertext Application Language (HAL) 349
- RESTful web services (REST) 349
- serialization 349

**MySQL my.cnf settings, Drupal installation**

- empty MySQL database, creating 13
- modifying 12
- modifying, in Mac OS X 12

modifying, in Windows 12

MySQL, setting up for Drupal 12

**N****NutritioninfoDefaultFormatter.php file**

references 105

**NutritioninfoDefaultWidget.php**

references 103

**NutritioninfoItem.php code**

references 99

**NutritionInformation module**

about 89

custom module, developing for compound

NutritionInformation field 90-105

Recipe content type, updating 106

references 89

**O****object 42****object-oriented programming (OOP) 42****OOP concepts, in Drupal**

about 42

abstraction 42

encapsulation 43

inheritance 45

object 42

polymorphism 44

**P****pager**

used, for enhancing front banner

appearance 274

**patch**

creating, Drupal 8/Git way 206, 207

uploading, on drupal issues queue 206, 207

**performance tuning tips**

reference 12

**PHP configuration, Drupal installation**

about 11

php.ini settings, modifying 11

php.ini settings, modifying in Mac OS X 11

php.ini settings, modifying in Windows 11

**phpMyAdmin**

for Mac OS X 13

for Windows 13

- PHPStorm IDE**  
about 21  
download link 21  
Drupalizing 23, 24  
installing 21, 22  
PHPStorm project, creating 22, 23  
reference 21
- PHPUnit tests for Drupal**  
reference 60
- polymorphism 44**
- PSR-4 namespaces**  
reference 49
- PuPHPet**  
URL 25
- R**
- random top rated recipe block**  
building, with views 226-230
- recipe listing block**  
creating, views used 66-70
- recipe reviews, with comments**  
about 208  
comments, configuring as recipe  
reviews 208-214  
liking system, enhancing with comments and  
views 214-223
- Representational State Transfer (REST)**  
about 344  
design principles 345
- RESTful APIs, Drupal**  
API, creating for exposing all recipes 357-361  
recipe types, obtaining 350-356  
RESTful web services, consuming with  
AngularJS 361-363
- rotating banners**  
building, with Views Slideshow module 266  
creating, with Views Slideshow module 267
- routes 48**
- routing file**  
reference 49
- routing system 48**
- S**
- sandbox project**  
promoting, to full project 283  
README.txt, creating 283, 284
- reference link 287  
Views semantic module, promoting to full  
project on Drupal.org 285-290
- search**  
entities, excluding from being indexed 314, 315  
exposing, to users 311, 312  
search display, altering 312, 313
- Search API Exclude Entity**  
reference 314
- Search API module**  
about 300, 301  
configuring 302  
fields 306, 307  
installing 301  
processors 307-309  
reference 300  
search indexes 303-305  
search index, populating 309, 310  
search servers 303, 304
- Search API Solr module**  
about 328  
Drupal, configuring to use Solr 328-331  
read-only mode, using 332
- Service Requester 344**
- Solr**  
configuring, on Ubuntu 324, 325  
installing, on Ubuntu 324, 325  
reference 324
- Solr 4.x installation**  
on virtual machine, with Vagrant  
and Puppet 316
- Solr 5.x**  
manual installation, on Ubuntu 14.04 323
- soup recipe 281, 282**
- sub-theme**  
creating 111, 112
- T**
- tabbed Views display 244, 245**
- Taxonomy-based view**  
creating, with tabs 232  
cuisine vocabulary, creating 232-234  
recipes, creating by cuisine type  
Views block 235-237
- Views Field View module, installing 239-243
- templates 126, 127**

- templating and Twig**  
about 126  
brackets syntax 130  
control structures 131  
file and function names 129  
filters 131  
references, for examples 128  
rendering 130, 131  
theme hook suggestions 128, 129
- test-driven development (TDD)**  
about 58, 59  
functional tests 60  
PHPUnit tests, for Drupal classes 60
- theme**  
about 109, 110  
sub-theme, creating 111, 112
- theming guide**  
reference 128
- Twig**  
features 129  
in practice 133-138  
reference 129
- Twig, debugging**  
about 132  
HTML comments, in markup 132  
Kint 133  
variables, debugging 132, 133
- U**
- Uncomplicated Firewall**  
configuring 327  
used, for securing Apache Solr 327
- Universally Unique Identifier (UUID) 73**
- V**
- Vagrant**  
about 25  
installing 25-28  
location, creating 25  
reference 25  
VM settings 26
- views**  
about 65, 66  
used, for creating recipe listing block 66-70
- Views configuration**  
advanced configuration 226
- Views Field View module**  
reference 239
- Views rotating banner**  
enhancing, with pager 274-281
- Views semantic tabs module**  
contributing, to Drupal 257  
sandbox, creating 258-262
- Views Slideshow module**  
installing 266  
new image style, creating for images 272-274  
used, for building rotating banners 265  
used, for creating rotating banners 267-271
- Views style plugin**  
developing, for semantic tabs 245-255
- VirtualBox**  
installing 25  
reference 25
- virtual machine**  
configuring 317-323  
creating, at PuPHPet 317-322
- W**
- web-based installation process**  
Mac OS X 19  
Windows 19
- web services 343, 344**
- X**
- XAMPP**  
reference 9
- Y**
- YAML 47**