PHP with Laravel for beginners - Become a Master in Laravel

Learn the Laravel Environment along with PHP and the MVC paradigm.

[Type the document subtitle]

# Contents

[Contents 1](#_Toc19092307)

[Section 1 - The First Steps 14](#_Toc19092308)

[1. Meet Your Professor Edwin 14](#_Toc19092309)

[2. Introduction To Laravel And MVC 14](#_Toc19092310)

[3. Reference Files 14](#_Toc19092311)

[4. Update (IMPORTANT please watch) 14](#_Toc19092312)

[Section 2 - Windows - Local Environment Setup 15](#_Toc19092313)

[5. Chapter Intro (IMPORTANT please watch) – Decommissioned – Content moved to lecture 4 15](#_Toc19092314)

[5. Intalling PHP Development Environment 15](#_Toc19092315)

[6. IDE Install 15](#_Toc19092316)

[7. Composer 15](#_Toc19092317)

[8. Git 15](#_Toc19092318)

[9. Laravel Version Update Information – Renamed – UPDATE – IMPORTANT – ABOUT ROUTES 15](#_Toc19092319)

[10. Creating A Laravel Project 15](#_Toc19092320)

[11. UPDATE - Dev Extensions Change – WIN 16](#_Toc19092321)

[12. Virtual Hosts 16](#_Toc19092322)

[13. Opening A Project In The IDE 16](#_Toc19092323)

[Section 3 - MAC- Local Environment Setup 17](#_Toc19092324)

[14. Section Intro With Edwin Diaz 17](#_Toc19092325)

[15. Installing PHP Development Environment 17](#_Toc19092326)

[16. IDE Install 17](#_Toc19092327)

[17. Composer 17](#_Toc19092328)

[18. UPDATE -IMPORTANT - ABOUT ROUTES & PERMISSIONS – MAC 17](#_Toc19092329)

[19. Creating A Laravel Project 17](#_Toc19092330)

[20. Course - UPDATE - App Dev Extensions – MAC 17](#_Toc19092331)

[21. Virtual Hosts 17](#_Toc19092332)

[22. Opening A Project In The IDE 17](#_Toc19092333)

[Section 4 - Laravel Fundamentals - Routes 18](#_Toc19092334)

[23. Laravel Structure Overview 18](#_Toc19092335)

[24. Route Introduction 19](#_Toc19092336)

[25. Routes part 1 19](#_Toc19092337)

[26. 26. Routes part 2 19](#_Toc19092338)

[27. Naming Routes 20](#_Toc19092339)

[Section 5 - Laravel Fundamentals - Controllers 21](#_Toc19092340)

[28. Intro to chapter with Edwin 21](#_Toc19092341)

[29. Creating Controllers 21](#_Toc19092342)

[30. Routing Controllers 23](#_Toc19092343)

[31. Passing data 23](#_Toc19092344)

[32. Resources and Controllers 24](#_Toc19092345)

[Section 6 - Laravel Fundamentals - Views 25](#_Toc19092346)

[33. Intro to chapter with Edwin 25](#_Toc19092347)

[34. Creating views and custom method 25](#_Toc19092348)

[35. Passing data to views 25](#_Toc19092349)

[Section 7 - Laravel Fundamentals - Laravel Blade templating engine 27](#_Toc19092350)

[36. Intro to chapter with Edwin 27](#_Toc19092351)

[37. Master layout setup 27](#_Toc19092352)

[38. Some more blade features 28](#_Toc19092353)

[Section 8 - Laravel Fundamentals - Database - Laravel Migrations 29](#_Toc19092354)

[39. Intro to chapter with Edwin 29](#_Toc19092355)

[40. Environment configurations 29](#_Toc19092356)

[41. Migrating 29](#_Toc19092357)

[42. Creating migration and dropping them 30](#_Toc19092358)

[43. Adding columns to existing tables using migrations 30](#_Toc19092359)

[44. Some more migration commands 31](#_Toc19092360)

[Section 9 - Laravel Fundamentals - Raw SQL Queries 32](#_Toc19092361)

[45. Section Intro with Edwin Diaz 32](#_Toc19092362)

[46. Inserting data 32](#_Toc19092363)

[47. Reading Data 32](#_Toc19092364)

[48. Updating Data 32](#_Toc19092365)

[49. Deleting Data 33](#_Toc19092366)

[Section 10 - Laravel Fundamentals - Database - Eloquent / ORM 34](#_Toc19092367)

[50. Section Intro with Edwin Diaz 34](#_Toc19092368)

[51. Reading Data 34](#_Toc19092369)

[52. Reading / Finding with Constraints 35](#_Toc19092370)

[53. More ways to retrieve data 35](#_Toc19092371)

[54. Inserting / Saving Data 35](#_Toc19092372)

[55. Creating data and configuring mass assignment 36](#_Toc19092373)

[56. Updating with Eloquent 36](#_Toc19092374)

[57. Deleting Data 37](#_Toc19092375)

[58. Soft Deleting / Trashing 37](#_Toc19092376)

[59. Retrieving deleted / trashed records 38](#_Toc19092377)

[60. Restoring deleted / trashed records 38](#_Toc19092378)

[61. Deleting a record permanently 39](#_Toc19092379)

[Section 11 - Laravel Fundamentals - Database - Eloquent Relationships 40](#_Toc19092380)

[62. Section intro with Edwin Diaz 40](#_Toc19092381)

[63. One to One relationship 40](#_Toc19092382)

[64. The inverse relation 41](#_Toc19092383)

[65. One to many relationship 41](#_Toc19092384)

[66. Some random tips 42](#_Toc19092385)

[67. Many to many relations part 1 42](#_Toc19092386)

[68. Many to many relations part 2 43](#_Toc19092387)

[69. Querying intermediate table 44](#_Toc19092388)

[70. Has many through relation part 1 45](#_Toc19092389)

[71. Has many through relation part 2 45](#_Toc19092390)

[72. Polymorphic relation part 1 46](#_Toc19092391)

[73. Polymorphic relation part 2 47](#_Toc19092392)

[74. Polymorphic relation the inverse 48](#_Toc19092393)

[75. Polymorphic relation many to many part 1 49](#_Toc19092394)

[76. Polymorphic relation many to many part 2 49](#_Toc19092395)

[77. Polymorphic relation many to many – retrieving 50](#_Toc19092396)

[78. Polymorphic relation many to many - retrieving owner 50](#_Toc19092397)

[Section 12 - Laravel Fundamentals - Database - Tinker 52](#_Toc19092398)

[79. Section Intro with Edwin Diaz 52](#_Toc19092399)

[80. Creating data with tinker 52](#_Toc19092400)

[81. Finding record and using constraints in tinker 54](#_Toc19092401)

[82. Updating and deleting with tinker 54](#_Toc19092402)

[83. Playing around with relations in tinker 55](#_Toc19092403)

[Section 13: Database - Eloquent One to One Relationship CRUD 58](#_Toc19092404)

[84. Section intro with Edwin Diaz 58](#_Toc19092405)

[85. Creating a new Laravel installation / Setup 58](#_Toc19092406)

[86. Database configuration and migration 58](#_Toc19092407)

[87. Setting up Relations 58](#_Toc19092408)

[88. Creating data for user 58](#_Toc19092409)

[89. Updating Data 59](#_Toc19092410)

[90. Reading and deleting data 59](#_Toc19092411)

[Section 14: Database - Eloquent One to Many Relationship CRUD 60](#_Toc19092412)

[91. Creating a new Laravel installation / Setup 60](#_Toc19092413)

[92. Database migrations 60](#_Toc19092414)

[93. Relationships and mass assignment 60](#_Toc19092415)

[94. Inserting Data 60](#_Toc19092416)

[95. Reading Data 61](#_Toc19092417)

[96. Updating Data 61](#_Toc19092418)

[97. Deleting Data 61](#_Toc19092419)

[Section 15: Database - Eloquent Many to Many Relationship CRUD 62](#_Toc19092420)

[98. Creating a new Laravel installation / Setup 62](#_Toc19092421)

[99. Database migrations 62](#_Toc19092422)

[100. Relationships and mass assignment 62](#_Toc19092423)

[101. Inserting Data 62](#_Toc19092424)

[102. Reading Data 63](#_Toc19092425)

[103. Updating Data 63](#_Toc19092426)

[104. Deleting Data 63](#_Toc19092427)

[105. Attaching, detaching and syncing 63](#_Toc19092428)

[Section 16: Database – Eloquent Polymorphic Relationship CRUD 65](#_Toc19092429)

[106. Creating a new Laravel installation / Setup 65](#_Toc19092430)

[107. Database migration 65](#_Toc19092431)

[108. Relationships and mass assignment 65](#_Toc19092432)

[109. Inserting Data 66](#_Toc19092433)

[110. Reading Data 66](#_Toc19092434)

[111. Updating Data 67](#_Toc19092435)

[112. Deleting Data 67](#_Toc19092436)

[113. Extra techniques 67](#_Toc19092437)

[Section 17: Database – Eloquent Polymorphic Many to Many Relationships 69](#_Toc19092438)

[114. Creating a new Laravel installation / Setup 69](#_Toc19092439)

[115. Database migrations 69](#_Toc19092440)

[116. Relationships and mass assignment 69](#_Toc19092441)

[117. Inserting Data 70](#_Toc19092442)

[118. Reading and updating data 70](#_Toc19092443)

[119. Deleting Data 70](#_Toc19092444)

[Section 18: Forms and Validation 72](#_Toc19092445)

[120. Section Intro with Edwin Diaz 72](#_Toc19092446)

[121. Setting up migration and relations 72](#_Toc19092447)

[122. Setting up views and routes 72](#_Toc19092448)

[123. 4-Setting the markup for create 73](#_Toc19092449)

[124. Setting up the controller and view 74](#_Toc19092450)

[125. Persisting data to database 74](#_Toc19092451)

[126. Reading data 75](#_Toc19092452)

[127. Showing individual posts and updating 75](#_Toc19092453)

[128. Preparing the edit view and setting up 76](#_Toc19092454)

[129. 10- Editing records and redirecting 77](#_Toc19092455)

[130. 11- Deleting 77](#_Toc19092456)

[Section 19: Forms - Package and Validation 78](#_Toc19092457)

[131. Section intro with Edwin Diaz 78](#_Toc19092458)

[132. UPDATE – IMPORTANT 78](#_Toc19092459)

[133. Installing package and testing 78](#_Toc19092460)

[134. Modifying our create form with the form package 79](#_Toc19092461)

[135. Modifying our edit and delete forms with the form package 80](#_Toc19092462)

[136. Basic validation 81](#_Toc19092463)

[137. Displaying errors 81](#_Toc19092464)

[138. Advance validation 82](#_Toc19092465)

[139. Creating snippets with PhpStorm 83](#_Toc19092466)

[Section 20: Database - Some more model manipulation 84](#_Toc19092467)

[140. Section Intro with Edwin Diaz 84](#_Toc19092468)

[141. Dates 84](#_Toc19092469)

[142. Accessors 84](#_Toc19092470)

[143. Mutators 85](#_Toc19092471)

[144. Query Scope 85](#_Toc19092472)

[Section 21: Forms - Uploading files 87](#_Toc19092473)

[145. Section intro with Edwin Diaz 87](#_Toc19092474)

[146. UPDATE – IMPORTANT – ABOUT UPLOADING 87](#_Toc19092475)

[147. Modifying our create view for file input 87](#_Toc19092476)

[148. Retrieving file data 87](#_Toc19092477)

[149. Persisting file data into the database 88](#_Toc19092478)

[150. Displaying images and using accessors to make it easy 89](#_Toc19092479)

[Section 22: Form - Login 90](#_Toc19092480)

[151. Section intro with Edwin Diaz 90](#_Toc19092481)

[152. New Laravel Installation 90](#_Toc19092482)

[153. Database connection and migrations 90](#_Toc19092483)

[154. Creating the login system under 1 minute 90](#_Toc19092484)

[155. Login creation overview 91](#_Toc19092485)

[156. Retrieving authenticated user data 91](#_Toc19092486)

[Section 23: Middleware - Security / Protection 93](#_Toc19092487)

[157. Section intro 93](#_Toc19092488)

[158. Create a new laravel installation (challenge) 93](#_Toc19092489)

[159. Registering a new middleware and using it 93](#_Toc19092490)

[160. A more practical way to use middlewares 1 - roles, migration a… 94](#_Toc19092491)

[161. A more practical way to use middlewares 2 - Custom method 95](#_Toc19092492)

[162. A more practical way to use middlewares 3 - Custom method 96](#_Toc19092493)

[163. A more practical way to use middlewares 4 - Custom method 96](#_Toc19092494)

[Section 24: Laravel Sessions 98](#_Toc19092495)

[164. Section intro with Edwin Diaz 98](#_Toc19092496)

[165. Setting and reading sessions 98](#_Toc19092497)

[166. Global session function deleting 98](#_Toc19092498)

[167. Flashing data 98](#_Toc19092499)

[Section 25: Laravel - Sending Email / Api 100](#_Toc19092500)

[168. Section intro with Edwin Diaz 100](#_Toc19092501)

[169. New laravel installation 100](#_Toc19092502)

[170. Overview on mailgun 100](#_Toc19092503)

[171. Email verification 100](#_Toc19092504)

[172. Configuration 100](#_Toc19092505)

[173. Sending email part 1 - route 101](#_Toc19092506)

[174. Sending email part 2 - Finished 101](#_Toc19092507)

[175. Custom domain 102](#_Toc19092508)

[Section 26: Git and Github – Version Control 104](#_Toc19092509)

[176. Git intro 104](#_Toc19092510)

[177. Git Installation 104](#_Toc19092511)

[178. Git commands 104](#_Toc19092512)

[179. Git crash lecture 104](#_Toc19092513)

[180. Git workflow 104](#_Toc19092514)

[181. Branches 105](#_Toc19092515)

[182. Creating repositories on Github 106](#_Toc19092516)

[183. Git settings 106](#_Toc19092517)

[184. Cloning and pushing to Github 106](#_Toc19092518)

[185. Pull request and pulling changes 108](#_Toc19092519)

[186. Contributing to the project 108](#_Toc19092520)

[Section 27: Application 109](#_Toc19092521)

[187. Laravel Project Reference Files 109](#_Toc19092522)

[188. New laravel installation 109](#_Toc19092523)

[189. Database configuration 109](#_Toc19092524)

[190. Setting up views 109](#_Toc19092525)

[191. Users table Migration 110](#_Toc19092526)

[192. Relation setup and data entry 111](#_Toc19092527)

[193. Testing relation with Tinker 111](#_Toc19092528)

[194. Admin controller and routes 112](#_Toc19092529)

[195. Testing methods 113](#_Toc19092530)

[196. Installing NodeJs and files download 113](#_Toc19092531)

[197. Gulp and assets 114](#_Toc19092532)

[198. Admin master file - download file 116](#_Toc19092533)

[199. Version control and modifying master page 117](#_Toc19092534)

[200. Displaying users 117](#_Toc19092535)

[201. Create page 118](#_Toc19092536)

[202. Laravel collective html package 119](#_Toc19092537)

[203. Testing form and creating form fields 119](#_Toc19092538)

[204. Working on user form status field 119](#_Toc19092539)

[205. Populating the user roles select 120](#_Toc19092540)

[206. Password field and custom request 120](#_Toc19092541)

[207. Displaying errors and including with blade 121](#_Toc19092542)

[208. Adding upload file feature to form 122](#_Toc19092543)

[209. Persisting data and fixing bug 122](#_Toc19092544)

[210. User photos migration - relation - mass-assignment 122](#_Toc19092545)

[211. Creating links and more 123](#_Toc19092546)

[212. Persisting user with photo part 1 124](#_Toc19092547)

[213. Persisting user with photo part 2 124](#_Toc19092548)

[214. Displaying photos using an accessor 125](#_Toc19092549)

[215. Edit users part 1 - setting up the form 125](#_Toc19092550)

[216. Edit user part 2 - displaying images and status 126](#_Toc19092551)

[217. Updating part 1 and displaying errors 127](#_Toc19092552)

[218. Updating part 2 127](#_Toc19092553)

[219. Updating part 3 - Fixing loose ends 128](#_Toc19092554)

[220. Security part 1 – middleware registration 129](#_Toc19092555)

[221. Security part 2 - middleware - custom method and 404 page 129](#_Toc19092556)

[222. Deleting users 130](#_Toc19092557)

[223. Some flash messages 131](#_Toc19092558)

[224. Deleting images from the directory 132](#_Toc19092559)

[225. Adding more security 132](#_Toc19092560)

[Section 28: Application - Posts 134](#_Toc19092561)

[226. Setting route files 134](#_Toc19092562)

[227. Migration 135](#_Toc19092563)

[228. Displaying post 135](#_Toc19092564)

[229. Relationship setup 137](#_Toc19092565)

[230. Creating form part 1 138](#_Toc19092566)

[231. Creating form part 2 138](#_Toc19092567)

[232. Creating form part 3 139](#_Toc19092568)

[233. Creating post 1 139](#_Toc19092569)

[234. Creating post 2 140](#_Toc19092570)

[235. Displaying images in post 140](#_Toc19092571)

[236. Creating model and migration for categories 140](#_Toc19092572)

[237. Displaying and creating posts with categories 141](#_Toc19092573)

[238. Let's stop and push to GitHub 141](#_Toc19092574)

[239. Pushing to GitHub warning 142](#_Toc19092575)

[240. Editing part 1 - setting up the page and form 143](#_Toc19092576)

[241. Editing part 2 - Lets edit the post 144](#_Toc19092577)

[242. Deleting - Challlenge 144](#_Toc19092578)

[243. Deleting - Solution 145](#_Toc19092579)

[244. Deleting users with relations 145](#_Toc19092580)

[245. Adding images to edit post 146](#_Toc19092581)

[Section 29: Application - Categories 147](#_Toc19092582)

[246. Setting up categories 147](#_Toc19092583)

[247. Creating categories 148](#_Toc19092584)

[248. Updating and deleting categories 149](#_Toc19092585)

[Section 30: Application - Media 152](#_Toc19092586)

[249. Setting up and displaying media 152](#_Toc19092587)

[250. Setting up the create view 153](#_Toc19092588)

[251. Adding the upload plugin 154](#_Toc19092589)

[252. Uploading images 155](#_Toc19092590)

[253. Deleting images 155](#_Toc19092591)

[Section 31: Application - Comments 157](#_Toc19092592)

[254. Setting up part 1 157](#_Toc19092593)

[255. Relations and mass assignment 158](#_Toc19092594)

[256. Setting up views 159](#_Toc19092595)

[257. Creating a post part 1 159](#_Toc19092596)

[258. Creating a post part 2 160](#_Toc19092597)

[259. Creating a post part 3 163](#_Toc19092598)

[260. Creating a comment 164](#_Toc19092599)

[261. Displaying posts 165](#_Toc19092600)

[262. Approving, un-approving and deleting part 1 - setting it up 166](#_Toc19092601)

[263. Approving, un-approving and deleting part 2 - Using it 167](#_Toc19092602)

[264. Displaying comments 167](#_Toc19092603)

[265. Displaying specific comments 169](#_Toc19092604)

[266. Displaying comments in post page 169](#_Toc19092605)

[267. Creating the reply form 171](#_Toc19092606)

[268. Creating a reply 171](#_Toc19092607)

[269. Displaying replies 172](#_Toc19092608)

[270. Displaying replies with JavaScript 174](#_Toc19092609)

[271. Displaying replies in admin part 1 175](#_Toc19092610)

[272. Displaying replies in admin part 2 177](#_Toc19092611)

[273. Updating and deleting replies 177](#_Toc19092612)

[Section 32: Extra Features 180](#_Toc19092613)

[*274.* Pretty url configuration part 1 180](#_Toc19092614)

[*275.* Pretty url configuration and finished part 2 - *Revisit after upgrading to laravel v.5.3* 180](#_Toc19092615)

[276. Laravel Pagination 180](#_Toc19092616)

[277. Pulling Gravatars from users 180](#_Toc19092617)

[Section 33: Upgrade to Laravel 5.3 182](#_Toc19092618)

[278. Brief introduction to changes 182](#_Toc19092619)

[279. Laravel project reference files download - version 5.3 182](#_Toc19092620)

[280. Upgrading - Extra folders in this version 182](#_Toc19092621)

[281. Upgrading routes part 1 186](#_Toc19092622)

[282. Upgrading routes part 2 - Login 188](#_Toc19092623)

[283. Upgrading routes part 3 - Route names 189](#_Toc19092624)

[284. Upgrading users 190](#_Toc19092625)

[285. Upgrading posts 191](#_Toc19092626)

[286. Upgrading categories 191](#_Toc19092627)

[287. Checking out media 192](#_Toc19092628)

[288. Upgrading middleware 192](#_Toc19092629)

[289. Upgrading our Pretty URL feature 192](#_Toc19092630)

[Section 34: Extra feature – WYSIWYG and File Management installation 195](#_Toc19092631)

[290. Part 1- Downloading packages and installing editor 195](#_Toc19092632)

['Image' => Intervention\Image\Facades\Image::class, 196](#_Toc19092633)

[291. Part 2- Making folders and uploading files 196](#_Toc19092634)

[292. Displaying images 197](#_Toc19092635)

[Section 35: Extra feature – Disqus Comment System 199](#_Toc19092636)

[293. Installing Disqus 199](#_Toc19092637)

[Section 36: Extra feature - Bulk Media Delete 200](#_Toc19092638)

[294. Part 1 - Creating form, method and route 200](#_Toc19092639)

[295. Part 2 - Testing our form and deleting 201](#_Toc19092640)

[296. Part 3 - Adding Javascript / jQuery 201](#_Toc19092641)

[297. Fixing bulk delete bug and new improvements 202](#_Toc19092642)

[298. Removing extra delete button 204](#_Toc19092643)

[Section 37: Extra feature: Deploying our app to a shared hosting account 205](#_Toc19092644)

[299. Intro 205](#_Toc19092645)

[300. SSH - WINDOWS OS 205](#_Toc19092646)

[301. SSH - MAC OS 205](#_Toc19092647)

[302. Module installation and PHP version 205](#_Toc19092648)

[303. Composer install 205](#_Toc19092649)

[304. Laravel installer 206](#_Toc19092650)

[305. Uploading project 206](#_Toc19092651)

[306. Configuring our environment file 206](#_Toc19092652)

[307. Our app live on the web 207](#_Toc19092653)

[308. Loging in 207](#_Toc19092654)

[309. Displaying images 208](#_Toc19092655)

[310. Deleting images 209](#_Toc19092656)

[311. Setting up production settings 209](#_Toc19092657)

[Section 38: Upgrade to Laravel 5.4 210](#_Toc19092658)

[312. Intro and part 1 210](#_Toc19092659)

[313. Laravel upgrade to 5.4 - part two 210](#_Toc19092660)

[314. Laravel upgrade to 5.4 - part three 212](#_Toc19092661)

[315. Webpack error - fix 212](#_Toc19092662)

[Section 39: Extra feature: Laravel data seeding 214](#_Toc19092663)

[316. Creating a simple seeder 214](#_Toc19092664)

[317. Creating a more advanced seeder with factories 215](#_Toc19092665)

[318. Let's create factories for all our database tables 216](#_Toc19092666)

[Section 40: Upgrade to Laravel 5.5 219](#_Toc19092667)

[319. Warning PHP Update for 5.5 - (Edwin from the future) 219](#_Toc19092668)

[320. Laravel Homestead Box Update (Edwin from the future) 219](#_Toc19092669)

[321. Upgrading the easy way 219](#_Toc19092670)

[Section 41: Upgrade to Laravel 5.6 220](#_Toc19092671)

[322. Upgrading to Laravel 5.6 220](#_Toc19092672)

[Section 42: Extra - Adjustments and more Features 221](#_Toc19092673)

[323. Making admin more modular 221](#_Toc19092674)

[324. Creating an admin controller 221](#_Toc19092675)

[325. Installing a Chart on admin 221](#_Toc19092676)

[326. Replacing Chart static data with dynamic data 221](#_Toc19092677)

[327. Some adjustments to admin posts page 221](#_Toc19092678)

[328. Installing a Blog Home Page 221](#_Toc19092679)

[329. Making front-end more modular 221](#_Toc19092680)

[330. Replacing static with dynamic data for home page 221](#_Toc19092681)

[331. Some adjustments to single post page 221](#_Toc19092682)

[332. Making front-end top navigation better 221](#_Toc19092683)

[333. Front-end Comment Flash messaging modification 221](#_Toc19092684)

[334. Pagination in the front-end 221](#_Toc19092685)

[Section 43: Upgrade from 5.6 to 5.7 & 5.7 to 5.8 222](#_Toc19092686)

[335. Let's start upgrading to 5.7 222](#_Toc19092687)

[336. Let's start upgrading to 5.8 222](#_Toc19092688)

[Appendix 223](#_Toc19092689)

[A. Special Note – Compatibility Issues 223](#_Toc19092690)

[B. Composer – Installing Laravel 223](#_Toc19092691)

[C. Listing Routes 223](#_Toc19092692)

[D. Creating Controllers 223](#_Toc19092693)

[E. Creating Migrations 223](#_Toc19092694)

[F. Creating Requests 223](#_Toc19092695)

[G. Implementing Migrations 223](#_Toc19092696)

[H. Creating Models 224](#_Toc19092697)

[I. Creating Login in Under 1 Minute 224](#_Toc19092698)

[J. Tinker 224](#_Toc19092699)

[K. File Locations 224](#_Toc19092700)

[L. Create Middleware 224](#_Toc19092701)

[M. GIT Commands 224](#_Toc19092702)

[N. Laravel Collective HTML Docs 225](#_Toc19092703)

[Forms & HTML 225](#_Toc19092704)

[Installation 225](#_Toc19092705)

[Opening A Form 226](#_Toc19092706)

[Form Model Binding 226](#_Toc19092707)

[CSRF Protection 228](#_Toc19092708)

[Labels 228](#_Toc19092709)

[Text, Text Area, Password & Hidden Fields 228](#_Toc19092710)

[Checkboxes and Radio Buttons 229](#_Toc19092711)

[Number 229](#_Toc19092712)

[Date 229](#_Toc19092713)

[File Input 229](#_Toc19092714)

[Drop-Down Lists 230](#_Toc19092715)

[Buttons 230](#_Toc19092716)

[Custom Macros 231](#_Toc19092717)

# Section 1 - The First Steps

## Meet Your Professor Edwin

## Introduction To Laravel And MVC

M - Model – Deals with the database

V - View – Deals with HTML

C - Controller – Middleman, directs requests to model or view

## Reference Files

See (in C:\Users\barry\Documents\Udemy\PHP with Laravel for beginners - Become a Master in Laravel):

codehacking-version-5.2.zip

codehacking-version-5.3.zip

codehacking-version-5.4.zip

codehacking-version-5.5.zip

codehacking-version-5.6.zip

routes.php.zip

## Update (IMPORTANT please watch)

Suggestion to look at the course Laravel Homestead. It’s free.

Talks about other free Laravel installers.

# Section 2 - Windows - Local Environment Setup

## 5. Chapter Intro (IMPORTANT please watch) – Decommissioned – Content moved to lecture 4

Using XAMPP in this course.

Laravel suggests using Laravel Homestead.

The professor offers a course in Laravel Homestead.

I’ve already got my own installed.

Alternate environment Laragon

## Intalling PHP Development Environment

Install XAMPP for PHP 7

Review how to change ports for Apache web server and SSL port.

## IDE Install

Install phpStorm

## Composer

Install Composer

Packagist.org is the site for Composer packages.

Tell Composer where to find php under XAMPP if one is not found.

## Git

Download, Install and configure Git

## Laravel Version Update Information – Renamed – UPDATE – IMPORTANT – ABOUT ROUTES

Start course in version 5.2

Updates made throughout this

Start updating the project in section 33 to v.5.3

See next lesson for the specific command to install a specific version of Laravel.

## Creating A Laravel Project

Open Git Bash

Go to HTDOCS folder

Command to create a new Laravel project;

Composer create-project --prefer-dist laravel/laravel ***{your project name}*** 5.2.29

-- or --

Composer create-project laravel/laravel ***{your project name}*** 5.2.31 --prefer-dist

Replace ***{your project name}*** with the name you want to give your project, in this case ***cms***

Under HTDOCS you will see a new directory named {your project name}, in this case cms or C:\wwwroot\tutorial\cms

## UPDATE - Dev Extensions Change – WIN

Warning .app and .dev domains have security issues (SSL) in the browser. Use .test instead.

## Virtual Hosts

To create a virtual host

Edit C:\Windows\System32\drivers\etc\hosts

Add the following;

127.0.0.1 www.tutorial.tst

127.0.0.1 tutorial.tst

Edit C:\Apache24\conf\extra\ httpd-vhosts.conf

Add the following paragraph;

<VirtualHost \*:80>

DocumentRoot "C:\wwwroot\tutorial"

ServerName www.tutorial.tst

ServerAlias tutorial.tst

ServerAdmin barry.rayfield@gmail.com

ErrorLog "logs/tutorial.tst-error.log"

CustomLog "logs/tutorial.tst-access.log" common

<Directory "C:\wwwroot\tutorial ">

Options Indexes FollowSymLinks

AllowOverride All

Require all granted

Options ExecCGI

</Directory>

</VirtualHost>

## Opening A Project In The IDE

For PHPStorm.

# Section 3 - MAC- Local Environment Setup

## Section Intro With Edwin Diaz

Going to install XAMPP on OS X

Starting with Laravel 5.2. Will ultimately upgrade to 5.7 as the class moves forward.

## Installing PHP Development Environment

Install and configure XAMPP on OS X

## IDE Install

Install PhpStorm

## Composer

Install Composer

Packagist.org

Go to your home directory and create .bash\_profile to add composer to the path

Command: source .bash\_profile

What it does: Refreshes your profile.

## UPDATE -IMPORTANT - ABOUT ROUTES & PERMISSIONS – MAC

Similar to Lecture 9 but for Mac instead of P.C.

## Creating A Laravel Project

See Windows section for install under Composer

## Course - UPDATE - App Dev Extensions – MAC

Warning about custom URL’s. Cannot use .dev or .app. This causes “Connection Not Private Error” message. Change it to .test or .tst instead. See Windows section for more information.

## Virtual Hosts

Modify httpd.conf to make sure the virtual hosts file in now included

Modify hosts file with new virtual hosts you are creating

Modify httpd-vhosts.conf to create the new hosts under apache

Add NameVirtualHost \*:80 above the virtual host definitions

Create the virtual hosts

## Opening A Project In The IDE

For PHPStorm

# Section 4 - Laravel Fundamentals - Routes

## Laravel Structure Overview

Application Folder Structure

C:\wwwroot\tutorial\cms.

***+---app***

| +---Console

| +---Events

| +---Exceptions

***| +---Http***

***| | +---Controllers controller classes***

***| | +---Middleware security***

***| | \---Requests***

***| | Kernal.php where you register security classes***

***| | Routes.php where you programmatically construct app URL’s***

| +---Jobs

| +---Listeners

| +---Policies

| \---Providers

+---bootstrap

***+---config***

***| app.php register classes from another plugin or provider***

***| database.php register database connections***

***| mail.php where you set up email configuration***

***+---database***

***| +---factories***

***| | ModelFactory.php classes used in laravel to create content***

***| +---migrations functionality provided to create tables with columns***

***| \---seeds***

+---nbproject

***+---public where we’ll store .css, .js files***

***| .htaccess***

***| favicon.ico***

***| index.php***

***| robots.txt***

***| web.config***

***+---resources***

***| +---assets***

***| | \---sass code that will get compiled to css***

| +---lang

***| \---views where we keep the views and markup***

***| +---errors***

***| \---vendor***

+---storage

+---tests

***\---vendor where all packages will be installed***

***.env Sets up environment variables for the application***

***.env.example template for .env***

***.gitattributes***

***.gitignore Gives instruction to git as to what should be sent back***

***up***

***artisan***

***composer.json required packages***

***composer.lock***

***gulpfile.js compile sass into css***

***package.json dependancies***

***phpunit.xml***

***readme.md***

***server.php***

## Route Introduction

Routes very important. It’s what’s in the URL in the address bar

## Routes part 1

Introduction to c:\wwwroot\tutorial\cms\app\Http\routes.php

This is the redirector for the application.

Core logic:

Route::get('/', function () {

return view('welcome');

});

***Route*** - classname

***::*** - use static function

***get()*** - name of static function

***‘/’*** - when you hit the application root url (in this case <http://tutorial.tst/cms/public/>)

***view()*** - display the view welcome. Stored in the view directory.

## 26. Routes part 2

Learned parameter passing

URL: <http://tutorial.tst/cms/public/>

Brings you to the Laravel welcome screen

**Code:** Route::get('/', function () {

return view('welcome');

});

URL: [http://tutorial.tst/cms/public/post](http://tutorial.tst/codehacking/public/post/1000/barry)

Brings you to a message that no id number was found

**Code:** Route::get('/post', function () {

return "You did not enter a post number ";

});

URL: [http://tutorial.tst/cms/public/post/1000](http://tutorial.tst/codehacking/public/post/1000)

Brings you to a message the id number entered in 1000

**Code:** Route::get('/post/{id}', function ($id) {

return "This is post number " . $id;

});

URL: [http://tutorial.tst/cms/public/post/1000/barry](http://tutorial.tst/codehacking/public/post/1000/barry)

Brings you to a message the id number entered is 1000 with a name of barry

**Code:** Route::get('/post/{id}/{name}', function ($id, $name) {

return "This is post number " . $id . “ “ . $name;

});

## Naming Routes

**Code:** Route::get('admin/posts/example', array('as'=>'admin.home', function () {

$url = route('admin.home');

return "This url is: " . $url;

}));

'admin/posts/example' Portion of URL after http://tutorial.tst/cms/public

array('as'=>'admin.home', function () {}

Create an associative array for 'admin/posts/example' as ‘admin.home’

$url = route('admin.home');

return "This url is: " . $url;

Retrieve the newly renamed route 'admin/posts/example' under the new name 'admin.home' the URL remain unmodified.

You can bring up a list of registered routes with the command php artisan route:list as shown below;

C:\wwwroot\tutorial\cms>php artisan route:list

+--------+----------+---------------------+------------+---------+------------+

| Domain | Method | URI | Name | Action | Middleware |

+--------+----------+---------------------+------------+---------+------------+

| | GET|HEAD | / | | Closure | web |

| | GET|HEAD | about | | Closure | web |

| | GET|HEAD | admin/posts/example | admin.home | Closure | web |

| | GET|HEAD | contact | | Closure | web |

| | GET|HEAD | post | | Closure | web |

| | GET|HEAD | post/{id} | | Closure | web |

| | GET|HEAD | post/{id}/{name} | | Closure | web |

+--------+----------+---------------------+------------+---------+------------+

Method – HTTP Request

Routes eliminate the need for some folders and html files.

For more information on Routes see;

<https://laravel.com/docs/5.2/routing>

# Section 5 - Laravel Fundamentals - Controllers

## Intro to chapter with Edwin

Controllers – business logic

Controllers take data from the models or views, process it then throw it back to the models for storage or views for display.

## Creating Controllers

Where are they located: {application root}\app\Http\Controllers

Namespace: Allows you to use same class names from different includes without causing collisions

Example: class RunThisQuery exist in two classes

Use: Import a specific namespace

Creating Controllers:

1. Standard way. File new, give it a name, etc.
2. Via Laravel:
   1. Open a terminal window in the IDE/Editor or open a CMD/Terminal Window.
   2. Make sure you’re in the applications root directory (C:\wwwroot\tutorial\cms)
   3. Type the following: php artisan make:controller PostController

The file PostController.php will be created in the directory app\Http\Controllers and will contain;

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class PostController extends Controller

{

//

}

* 1. Alternate method: Type php artisan make:controller --resource PostController

The file PostController.php will be created in the same location but will contain templating for the basic CRUD methods. For example;

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class PostController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

//

}

/\*\*

\* Show the form for creating a new resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function create()

{

//

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

//

}

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id)

{

//

}

/\*\*

\* Show the form for editing the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function edit($id)

{

//

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, $id)

{

//

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function destroy($id)

{

//

}

}

## Routing Controllers

Entering php artisan in the command window gives you a list of the available commands.

In Routes.php you direct a request to a controller with the following syntax;

Route::get('/posts', 'PostsController@index');

'PostsController@index' = {Controller} @ {Method}

## Passing data

Url is: <http://tutorial.tst/cms/public/posts/5>

Route is: Route::get('/posts/{a}', 'PostsController@index');

Controller syntax is:

public function index($id)

{

//

return "It's working. recieved an id of:" . $id;

}

Variable name do not have to be in sync.

The URL can have more variables that the method but not the reverse

Good:

URL: <http://tutorial.tst/cms/public/posts/500/100>

Route: Route::get('/posts/{a}/{b}', 'PostsController@index');

Controller Method Header: public function index($id)

Bad:

URL: <http://tutorial.tst/cms/public/posts/500/100>

Route: Route::get('/posts/{a}', 'PostsController@index');

Controller Method Header: public function index($id, $age)

To Fix Bad:

Set an initialization for the additional variables.

Controller Method Header: public function index($id, $age = 0)

## Resources and Controllers

For more information see: <https://laravel.com/docs/5.2/controllers>

Resource automatically creates routes for you.

Example; Route::resource('posts', 'PostsController');

Will automatically create;

C:\wwwroot\tutorial\cms>php artisan route:list

+--------+-----------+--------------------+---------------+------------------------+------------+

| Domain | Method | URI | Name | Action | Middleware |

+--------+-----------+--------------------+---------------+------------------------+------------+

| | GET|HEAD | posts | posts.index |App\Http\Controllers\ | web |

| | | | | PostsController@index | |

| | POST | posts | posts.store |App\Http\Controllers\ | web |

| | | | | PostsController@store | |

| | GET|HEAD | posts/create | posts.create |App\Http\Controllers\ | web |

| | | | | PostsController@create| |

| | GET|HEAD | posts/{posts} | posts.show |App\Http\Controllers\ | web |

| | | | | PostsController@show | |

| | PUT|PATCH | posts/{posts} | posts.update |App\Http\Controllers\ | web |

| | | | | PostsController@update| |

| | DELETE | posts/{posts} | posts.destroy |App\Http\Controllers\ | web |

| | | | | PostsController@destroy| |

| | GET|HEAD | posts/{posts}/edit | posts.edit |App\Http\Controllers\ | web |

| | | | | PostsController@edit | |

+--------+-----------+--------------------+---------------+------------------------+------------+

Method – HTTP Request

Action – Class name and method that is acted upon

Regardless of what the controller actually contains the default syntax for Route::resource() will always create the seven routes shown above.

You can limit the routes created with the “only” and “except” clauses of Route::resource(). See <https://laravel.com/docs/5.8/controllers#resource-controllers> and look for “Partial Resource Routes”

# Section 6 - Laravel Fundamentals - Views

## Intro to chapter with Edwin

Welcome to the Views section

## Creating views and custom method

Located at: {application root}\resources\views

Modification to Routes.php: None. No direct changes

Modification to PostsController.php: return view('contact');

Added to the method being processed.

If the template is stored in a subdirectory under view it can be accessed with the syntax:

view('{subdirectory}***\***contact');

or

view('{subdirectory}***.***contact');

Use of “\” or “/” seems to be OS specific. Use DIRECTORY\_SEPARATOR instead.

The preferred directory separator is the period “.”. It should always be used in place of “\” or “/”.

By using the “.” Laravel will take care of inserting the correct directory separator.

If using blade as the templating engine the file naming convention is: contact.blade.php

## Passing data to views

For more information see: <https://laravel.com/docs/5.2/views>

Passing a single parameter to a view;

**Route.php:** Route::get('post/{id1}', 'PostsController@show\_post');

**PostsController.php:**

public function show\_post($id1) {

return view('post')->with('id', $id);

}

**Post.blade.php:**

<!DOCTYPE html>

<html>

<head>

<title>Contact</title>

<link href="https://fonts.googleapis.com/css?family=Lato:100" rel="stylesheet" type="text/css">

</head>

<body>

<div class="container">

<h1>Post Page</h1>

<h2>This is id1: {{$id1}}</h2>

</div>

</body>

</html>

Passing a single parameter to a view;

**Route.php:** Route::get('post/{id1}/{id2}/{id3}', 'PostsController@show\_post');

**PostsController.php:**

public function show\_post($id1, $id2, $id3) {

return view('post', compact('id1', 'id2', 'id3'));

}

**Post.blade.php:**

<!DOCTYPE html>

<html>

<head>

<title>Contact</title>

<link href="https://fonts.googleapis.com/css?family=Lato:100" rel="stylesheet" type="text/css">

</head>

<body>

<div class="container">

<h1>Post Page</h1>

<h2>This is id1: {{$id1}}</h2>

<h2>This is id2: {{$id2}}</h2>

<h2>This is id3: {{$id3}}</h2>

<h2>Again this is id1: {{$id1}}</h2>

<h2>Again this is id2: {{$id2}}</h2>

<h2>Again this is id3: {{$id3}}</h2>

</div>

</body>

</html>

# Section 7 - Laravel Fundamentals - Laravel Blade templating engine

## Intro to chapter with Edwin

Blade – php templating engine

## Master layout setup

No changes to Route.php

No changes to controller

Best Practice: create a layouts directory under views directory. This is where the master “skeletons” reside for the various pages.

Best Practice: name of the master layout is usually app.blade.php

***layout/app.blade.php:***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

**@yield('content')**

</div>

**@yield('footer')**

</body>

</html>

@yield({name}) is a section that will be substituted for by the view.

***contact.blade.php***

@extends('layouts.app') //tells blade to use layouts/app as the layout master

@section('content') //tells the master (layouts/app) the following substitutes @yield(‘content’)

<h1>Contact Page</h1> //content that substitutes @yield(‘content’)

@stop //end of the @yield(‘content’) section

@section('footer') //tells the master (layouts/app) the following substitutes @yield(‘footer’)

<script>alert('Hello Visitor')</script>

//content that substitutes @yield(‘footer’)

@stop //end of the @yield(‘footer’) section

***post.blade.php***

@extends('layouts.app') //tells blade to use layouts/app as the layout master

@section('content') //tells the master (layouts/app) the following substitutes @yield(‘content’)

<h1>Posts Page</h1> //content using the passed values

<h2>id1 = {{$id1}}</h2>

<h2>id2 = {{$id2}}</h2>

<h2>id3 = {{$id3}}</h2>

@stop //end of the @yield(‘content’) section

//no need for footer therefore not passed

## Some more blade features

For more information see: <https://laravel.com/docs/5.2/blade>

Display an array of data

Modified PostsController.php

public function contact()

{

$people = ['edwin', 'jose', 'james', 'Peter', 'Maria'];

return view('contact', compact('people'));

}

Modified contact.blade.php

<h1>Contact Page</h1>

***@if*** (count($people))

<ul>

***@foreach***($people as $person)

<li>{{$person}}</li>

***@endforeach***

</ul>

***@endif***

# Section 8 - Laravel Fundamentals - Database - Laravel Migrations

## Intro to chapter with Edwin

What is a migration? Define a table in php and laravel will define it in the backend database.

Done by php artisan command

## Environment configurations

.env – environment variables that define the environment

.env located – {application root}\.env

.env is skipped by Github. See {application root}\.gitingore file.

env('DB\_DATABASE', 'forge')

env() - function for obtaining value of environment variables

'DB\_DATABASE' – name of the environment variable to read

‘forge’ – default value returned if environment variable does not exist

To see .env and env() used for the database environment see {application root}\config\database.php

For more info on .env and the concept see <https://github.com/vlucas/phpdotenv>

Typical laravel database config is for connection to one database. For info on connection to more than one dtabase at a time see;

<https://stackoverflow.com/questions/31847054/how-to-use-multiple-database-in-laravel>

<http://laraveldaily.com/multiple-database-connections-in-the-same-laravel-project/>

Official docs: <https://laravel.com/docs/5.4/database#using-multiple-database-connections>

## Migrating

Migration is a feature of Laravel that assists you in creating tables in the database.

Files supporting migration are located in {application root}\database\migrations

Contains classes for defining the tables in the database.

Dependency injection: <https://coderoncode.com/dependency-injection/design-patterns/programming/php/development/2014/01/06/dependency-injection-php.html>

For more information on the database migration utility: <https://laravel.com/docs/5.8/migrations>

The creation step is launched via php artisan.

Example run using the sample files;

C:\wwwroot\tutorial\cms>php artisan migrate

Migration table created successfully.

Migrated: 2014\_10\_12\_000000\_create\_users\_table

Migrated: 2014\_10\_12\_100000\_create\_password\_resets\_table

## Creating migration and dropping them

Create migration from scratch;

Command: php artisan make:migration create\_posts\_table --create=”posts”

create\_posts\_table: Partial output filename. It will be prefixed by date and time of file generation. It will be suffixed by .php

“posts”: Name of table to be created. It will be embedded in the class.

Created file: {application root}\cms\database\migrations\ 2019\_05\_21\_151559\_create\_posts\_table.php

Command: php artisan migrate

Result: Posts table is created

To delete the last batch of tables created:

Command: php artisan migrate:rollback

Result: The last “batch” of tables created will be dropped. To see how the tables are batched see the migrations table in the database.

## Adding columns to existing tables using migrations

Create migration to modify a table:

Command: php artisan make:migration add\_is\_admin\_column\_to\_posts\_table --table="posts"

add\_is\_admin\_column\_to\_posts: Partial output filename. It will be prefixed by date and time of file generation. It will be suffixed by .php

“posts”: Name of table to be modified. It will be embedded in the class.

Created file: {application root}\cms\database\migrations\ 2019\_05\_21\_174742\_add\_is\_admin\_column\_to\_posts\_table.php

The classes in this file use the method Schema::table to add and drop columns as opposed to Schema::create or Schema::drop as was used in creating the table.

Command: php artisan migrate

Result: Posts table is created

## Some more migration commands

For more information see: <https://laravel.com/docs/5.2/migrations>

Remove all that was created with migration: php artisan migrate:reset

Reapply all migrations: php artisan migrate

php artisan migrate:refresh Does a rollback and a migrate in one shot

php artisan migrate:status Gives a listing of what migrations have been applied

Info on database seeding: <https://laravel.com/docs/5.8/seeding>

# Section 9 - Laravel Fundamentals - Raw SQL Queries

## Section Intro with Edwin Diaz

Intro

## Inserting data

Raw SQL queries

Doing it the hard way

Lookup façade: The Facade Pattern (or façade pattern) is a software design pattern commonly used with object-oriented programming. The name is by analogy to an architectural facade. A facade is an object that provides a simplified interface to a larger body of code, such as a class library.

The DB class itself presents the facade pattern.

Code:

DB::insert(

'insert into posts (title, content) values (?, ?)',

['PHP with Laravel', 'Laravel is the best thing that happened to PHP']

);

Using the Model

Comes with defines classes for create (insert), read (select), update and delete.

See Eloquent Model

## Reading Data

Read (select)

Code:

$results = DB::select('select \* from posts where id = ?', [2]);

foreach ($results as $post) {

return var\_dump($post);

}

## Updating Data

Update (update)

Code:

Route::get('/update', function () {

$updated = DB::update(

'update posts set title = "update title" where id = ?',

[2]

);

return 'Number of rows updated: ' . $updated;

});

The update statement returns the number of rows affected not a result set. If the example above, $updated will be set to 1 if successful.

## Deleting Data

Delete (delete)

Code

Route::get('/delete', function () {

$deleted = DB::delete('delete from posts where id = ?', [2]);

return 'Number of rows deleted: ' . $deleted;

});

Just like with the update statement, delete returns the number of rows affected and not a result set. If the example above, $deleted will be set to 1 (the number of records deleted) if successful.

For more information: <https://laravel.com/docs/5.2/database>

# Section 10 - Laravel Fundamentals - Database - Eloquent / ORM

## Section Intro with Edwin Diaz

Intro

## Reading Data

To get Laravel to create a model utilizing Eloquent use the following command

php artisan make:model Post

Post – name of the class and the filename

This will create the file {application root}\app\Post.php with all the plumbing to use Eloquent.

Code:

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model

{

//

}

In its default settings Eloquent expects the class name to be singular while the table name is plural. For example:

Post – Class name, file name and record

Posts – database table name

If the table name were something else, such as “blog” you would need to add a protected property to the body of the Post class. For example;

protected $table = ‘blog’;

This informs Eloquent of the table name change.

Another expectation of Eloquent is for there to be a primary key field called ‘id’. If the your primary key field is called something else you can inform Eloquent of this with the protected property ‘$primaryKey’. For example:

protected $primaryKey = ‘post\_id’;

Routes:

Reading all records from the table (select \* from posts)

Route::get('/readall', function () {

$display = '';

$posts = Post::all();

foreach ($posts as $post) {

$display .= 'title: ' . $post->title . ' comment: ' . $post->content

. '<br>';

}

return $display;

});

Reading a specific record from the table (select \* from posts where id = 5)

Route::get('/read', function () {

$post = Post::find(5);

return 'title: ' . $post->title . ' comment: ' . $post->content . '<br>';

});

## Reading / Finding with Constraints

Creating a where clause on any field and not just the id field;

$posts = Post::***where('id', 6)***->orderBy('id', 'desc')->take(1)->get();

where('id', 6) – creates the clause “where id = 6”.

orderBy('id', 'desc') – select the column to sort on and ascending or descending.

take(1) – Select only the first record.

## More ways to retrieve data

Code:

$posts = Post::findorfail(1);

findorfail() – will return records if found otherwise generate a catchable 404 exception, page not found

1 – id that doesn’t exist

Code:

$posts = Post::where(‘users\_count’, ‘<’, 50)->firstOrFail();

where(‘users\_count’, ‘<’, 50) – Generate the clause ‘where users\_count < 50’

firstOrFail() – Return first record that meets the criteris. If none meet the criteria generate a 404 exception, page not found.

## Inserting / Saving Data

Eloquent / ORM – ORM stands for Object Relational Mapping, which is a programming technique that helps in converting data between incompatible type systems into object-oriented programming languages. Laravel uses "Eloquent ORM" which allows you to work with your database objects and relationships using an eloquent and expressive syntax.

The following code creates a brand new record (insert) in the database.

Code:

Route::get('/basicinsert', function () {

$post = new Post(); **//signifies insert**

$post->title = 'New Eloquent Title Insert again';

$post->content = 'Now Eloquent really is cool, look at ‘ .

‘this content.updated';

$post->save();

});

The following code does an update of an existing record. It works by finding the record first, instead of ‘new Post()’

Code:

Route::get('/basicinsert2', function () {

$post = Post::find(6); **//find the reord to be updated**

$post->title = 'Updated Title D'; **//make updates**

$post->save(); **//save actually update it**

});

## Creating data and configuring mass assignment

This involves using the create method

If you do not prepare the model (Post.php) to use the create method you get the error;

MassAssignmentException in Model.php

Preparation involves adding the following statement to the model (Post.php):

protected $fillable = [ **//$fillable is an array**

'title', **//list of updatable columns**

'content'

];

Now add the following to Route.php:

Post::create(

[

'Column 1'=>'Value 1',

'Colunm 2'=>'Value 2']

);

If create has less columns than those defined in $fillable, that's fine. If create has more meaning some columns in create that are not defined in $fillable, it will throw an error.

The sequencing of the columns in create and $fillable does not matter, just no new columns in create.

***To create multiple records you need to use the createMany() method.***

## Updating with Eloquent

Use of the update method.

Code:

Post::where('id', 5)->where('is\_admin', 0)->***update***(

['title'=>'NEW PHP TITLE', 'content'=>'I love my instructor Edwin']);

This chains multiple conditions on to the update (id = 5 and is\_admin = 0).

## Deleting Data

To delete an instantiated row use the delete() method.

Code:

Route::get('/delete', function () {

$post=Post::find(3);

$post->delete();

});

$post=Post::find(3); - Find and instantiate the record

$post->delete(); - Delete the instantiated record

To delete when you know the key of the record.

Code:

Route::get('/delete2', function () {

Post::destroy(4);

});

Post::destroy(4); - Delete the record with an id (key) of 4.

To delete multiple when you know the keys.

Code:

Route::get('/delete3', function () {

Post::destroy(4, 5);

});

Post::destroy(4, 5); - Delete records with the ids (keys) 4 and 5

## Soft Deleting / Trashing

To use soft deletes you need to create and apply a migration to create the deleted\_at column.

The default value for deleted\_at is NULL

To use it both the model (Post.php) and the route (routes.php) have to be modified.

Modifications to Post.php

class Post extends Model

{

***use SoftDeletes;*** //enables use of the softdelete

***protected $dates = ['deleted\_at'];*** //inform laravel that deleted\_at is a date column

protected $fillable = ['title', 'content'];

}

use SoftDeletes; - Enables use of the Soft Delete class, overriding the default delete method

protected $dates = ['deleted\_at']; - Inform Laravel that deleted\_at is a date column

Modifications to routes.php

Route::get('/softdelete', function () {

***Post::find(14)->delete();***

});

Post::find(14)->delete(); - finds record with an id of 14 and the does a soft delete

Soft delete consists of loading current date and time into the deleted\_at column.

When Laravel see the deleted\_at column does not contain a NULL value it ignores it for follow on queries.

Issue with softdelete:

If you have any columns that are flagged as unique in the database then new record creation will fail if the new record has any of the same keys as the softdeleted records.

## Retrieving deleted / trashed records

Simple find() will ignore records that have been soft deleted.

Code:

Route::get('/readsoftdelete', function () {

$post = Post::find(14);

return $post;

});

Returns nothing.

The method withTrashed() will add back the softdeleted records in to a query.

Code:

Route::get('/readsoftdelete', function () {

$post = Post::withTrashed()->where('id', 14)->get();

return $post;

});

Returns the softdeleted record in the result set.

The method onlyTrashed will query only those records that have been softdeleted.

Code:

Route::get('/readallsoftdelete', function () {

$post = Post::onlyTrashed()->get();

return $post;

});

Returns only those that meet the criteria and have been softdeleted.

## Restoring deleted / trashed records

The restore() method is used to reverse a softdelete (make deleted\_at a null value once more).

Code:

Route::get('/restore', function () {

Post::onlyTrashed()->find(15)->restore();

});

## Deleting a record permanently

The method forceDelete() is used to actually delete a record, soft deleted or not. You need to filter the record select properly.

Record with id of 15 was softdeleted. This permanently deletes the record

Code:

Route::get('/forcedelete', function () {

Post::***onlyTrashed()->find(15)***->forceDelete();

});

Record with id of 19 was not softdeleted. This forces it’s permanent deletion.

Code:

Route::get('/forcedelete', function () {

Post::***withTrashed()->find(19)***->forceDelete();

});

This forces the permanent deletion of all softdeleted records.

Code:

Route::get('/forcedelete', function () {

Post::onlyTrashed()->forceDelete();

});

For more information see: <https://laravel.com/docs/5.2/eloquent>

***Transaction Processing*** is not covered in this course. For more information see:

<https://stackoverflow.com/questions/22906844/laravel-using-try-catch-with-dbtransaction>

<https://laravel.com/docs/5.8/database#database-transactions>

Google “transaction processing with laravel”

<https://www.google.com/search?q=transaction+processing+with+laravel&rlz=1C1JZAP_enUS711US711&oq=transaction+processing+with+laravel&aqs=chrome..69i57.9052j0j7&sourceid=chrome&ie=UTF-8>

# Section 11 - Laravel Fundamentals - Database - Eloquent Relationships

## Section intro with Edwin Diaz

Overview of what’s coming: Database Relationships.

## One to One relationship

Add user\_id column to posts as unsigned integer.

Set one of the posts to have a user\_id of the new user (“1”).

Setup for the query:

Modify users and add this method to model Users.php

public function post() {

return $this->hasOne('App\Post');

}

hasOne() – does a lookup from the parent to the child. Expects one parent and will only return one child.

'App\Post' – is the model (table) that has one child. If there are multiple children only one child will be returned.

{user\_id column} – ***Not shown.*** The 2nd parameter is user id column in the child table is expected to be named user\_id. If it is it is not necessary to pass it. If the column is named something else it must be passed and enclosed in quotes.

{local key column} – ***Not shown.*** The 3rd parameter is id column in the parent table it is expected to be named id. If it is it is not necessary to pass it. If the column is named something else it must be passed and enclosed in quotes.

Set up the route:

Modify routes.php with the following:

Add the following at the top of routes.php to import the users model:

use App\User;

Add the route:

Route::get('/user/{id}/post/', function ($id) {

return User::find($id)->post;

});

Access with the URL:

<http://tutorial.tst/cms/public/user/1/post>

Result:

Display one or first post with a user id of 1.

If there are no posts you will see a blank page.

If userid 1 does not exist in the user table you will get a page does not exist message.

## The inverse relation

Lookup the parent of a child.

Uses the belongsTo() method

Format:

belongsTo({parent model}, {child model key field}, {parent model key field})

{parent model} – Location and name of the model that contains the parent definition.

{child model key field} – Name of the column in the child model that contains the foreign key. Only necessary if deviates from default.

{parent model key field} – Name of the column in the parent model that contains the primary key. Only necessary if deviates from default.

Modification made to Post.php:

public function user(){

return $this->belongsTo('App\User');

}

Using the currently retrieved post retrieve the parent user record based on the default child record (Post) column name (foreign key) of user\_id and the default parent record (User) column name (primary key) of id.

Modification to routes.php:

Add the following at the top of routes.php to import the Posts model:

use App\Post;

Add the following route;

Route::get('/post/{id}/user/', function ($id) {

return Post::find($id)->user->name;

});

## One to many relationship

One parent many children

Uses the hasMany() method

Format:

hasMany({child model}, 'foreign\_key', 'local\_key');

{child model} – Location and name of the model that contains the children definition.

{foreign key} – Name of the column in the child model that contains the foreign key. Only necessary if deviates from default.

{local key} – Name of the column in the parent model that contains the primary key. Only necessary if deviates from default.

Modification made to User.php:

public function posts()

{

return $this->hasMany('App\Post');

}

Using the currently retrieved user retrieve the children post records based on the default child record (Post) column name (foreign key) of user\_id and the default parent record (User) column name (primary key/local key) of id.

Modification to routes.php:

Add the following at the top of routes.php to import the Posts model:

use App\User;

Add the following route;

Route::get('/posts/{id}', function ($id) {

$user=User::find($id);

foreach ($user->posts as $post) {

echo $post->title . '<br>';

}

});

## Some random tips

Instructor ran into a problem where php artisan stopped working.

His solution was to delete the created tables and then try rerunning the migrations. Export data first?

## Many to many relations part 1

User table

Roles table

php artisan make:model Role –m

Created both model and migration.

Layout:

Schema::create('roles', function (Blueprint $table) {

$table->increments('id');

$table->string('name');

$table->timestamps();

});

Pivot table

php artisan make:migration create\_users\_roles\_table --create=role\_user

Created only the migration.

Layout:

Schema::create('role\_user', function (Blueprint $table) {

$table->increments('id');

$table->integer('user\_id');

$table->integer('role\_id');

$table->timestamps();

});

## Many to many relations part 2

By use of the Laravel method belongsToMany(), Laravel understands the use of a pivot table (in this case role\_user) comes into play.

Modifications to User.php (only needed to bring up the roles for a specific user):

Request for all role belonging to the current user.

public function roles()

{

return $this->belongsToMany('App\Role');

}

Modifications to Role.php (only needed to bring up the users assigned to a specific role):

Request for all users belonging to the current role.

public function users()

{

return $this->belongsToMany('App\User');

}

Modifications to routes.php:

Given the User {id} list all the roles

Route::get('/user/{id}/role', function ($id) {

// $user=User::find($id)->roles()->orderBy('id', 'desc')->get();

$user=User::find($id);

echo 'User: ' . $user . '<br>';

foreach ($user->roles as $role) {

echo 'Role: ' . $role->name . '<br>';

}

});

Given the role {id} list all the users

Route::get('/role/{id}/user', function ($id) {

//$role=Role::find($id)->users()->orderBy('id', 'desc')->get();

$role=Role::find($id);

echo 'Role: ' . $role . '<br>';

foreach ($role->users as $user) {

echo 'User: ' . $user->name . '<br>';

}

});

## Querying intermediate table

belongsToMany() Syntax:

$this->belongsToMany('App\User', 'role\_user', 'user\_id', 'role\_id');

'App\User' – Other model in the many to many relationship. Required.

'role\_user' – Name of the many to many table. Only needed if you deviate from the Laravel naming conventions.

'user\_id' – Name of the column in the Other model to key off of. Only needed if you deviate from the Laravel naming conventions.

'role\_id' – Name of the column in the current model to key off of. Only needed if you deviate from the Laravel naming conventions.

To query intermediate table create the following route:

Route::get('user/pivot', function () {

$user = User::find(3);

foreach ($user->roles as $role) {

// echo $role->pivot;

echo 'user\_id: ' . $role->pivot->user\_id .

' role\_id: ' . $role->pivot->role\_id .

' created\_at: ' . $role->pivot->created\_at .

'<br>';

}

});

And User.php with the following:

public function roles()

{

return $this->belongsToMany('App\Role')->withPivot('created\_at');

}

By default, only the model keys will be present on the pivot object. If your pivot table contains extra attributes, you must specify them when defining the relationship.

If you want your pivot table to have automatically maintained created\_at and updated\_at timestamps, use the withTimestamps method on the relationship definition:

return $this->belongsToMany('App\Role')->withTimestamps();

## Has many through relation part 1

The "has-many-through" relationship provides a convenient short-cut for accessing distant relations via an intermediate relation. For example, a Country model might have many Post models through an intermediate User model.

Prep for “has-many-through” relationships

Create 2 migrations:

php artisan make:model Country –m

Create a new table called countries:

***# Column Name Type***

1 id int(10) Primary Key

2 name varchar(255)

3 created\_at timestamp

4 updated\_at timestamp

php artisan make:migration add\_country\_id\_column\_to\_users --table=users

Add a new column to user table called country\_id

***# Column Name Type***

1 country\_id int(10) Primary Key

Populate countries table with:

1 DR

2 PR

3 Russia

4 US

5 India

6 Israel

7 Canada

8 U.K.

## Has many through relation part 2

Coding changes:

Routes.php

use App\Country; ***// at the top***

Route::get('/user/country', function () {

$country = Country::find(1);

echo 'country: id:', $country->id,

' name: ', $country->name,

' created\_at: ', $country->created\_at,

' updated\_at: ', $country->updated\_at,

'<br>';

echo var\_dump($country), '<br>';

foreach ($country->posts as $post) {

echo 'Post: Id: ', $post->id,

' Title: ', $post->title,

' Content: ', $post->content,

' created\_at: ', $post->created\_at,

' updated\_at: ', $post->updated\_at,

'<br>';

echo var\_dump($post), '<br>';

}

});

Country.php

public function posts()

{

return $this->hasManyThrough('App\Post', 'App\User');

}

Full syntax for hasManyThrough():

hasManyThrough({final model}, {intermediate model},

{foreign key on the intermediate model},

{foreign key on the final model}, {local key})

final model – Required. The final model in the chain begun by the current model (class).

intermediate model – Required. The final model in the chain begun by the current model (class).

foreign key on the intermediate model – Optional. Only needed if you deviated from standard naming conventions.

foreign key on the final model – Optional. Only needed if you deviated from standard naming conventions.

local key – Optional. Only needed if you deviated from standard naming conventions.

## Polymorphic relation part 1

Polymorphic relations allow a model to belong to more than one other model based on a single association.

Preparation is to create the migration and model for the photo table.

php artisan make:model Photo –m

Generated structure looks like;

1 idPrimary int(10) UNSIGNED Primary

2 path varchar(255)

3 imageable\_id int(11)

4 imageable\_type varchar(255)

5 created\_at timestamp

6 updated\_at timestamp

## Polymorphic relation part 2

A polymorphic relationship is where a model can belong to more than one other model on a single association.

Example: A comments table can hold comments for Topics and a Posts model (tables).

This is done by embedding a key (id) and type (column will contain the class name of the owning model) columns in the polymorphic table.

Coding changes made:

Photos.php:

public function imageable()

{

return $this->morphTo();

}

morphTo() - Define a polymorphic, inverse one-to-one or many relationship.

Photos model holds photos for users and posts. Signified by the existence of the imageable \_id and imageable\_type columns.

Designates a user photo:

imageable\_id – contains a valid user\_id

imageable\_type – contains the model name ‘App\User’

Designates a post photo:

imageable\_id – contains a valid post\_id

imageable\_type – contains the model name ‘App\User’

Posts.php

public function photos()

{

return $this->morphMany('App\Photo', 'imageable');

}

morphMany() - Define a polymorphic one-to-many relationship.

'App\Photo' – Indicates who has the relationship.

'imageable' – Indicates the root fir the \_id and \_type column names.

Users.php

public function photos()

{

return $this->morphMany('App\Photo', 'imageable');

}

morphMany() - Define a polymorphic one-to-many relationship.

'App\Photo' – Indicates who has the relationship.

'imageable' – Indicates the root fir the \_id and \_type column names.

Routes.php

Route::get('user/photos', function () {

$user = User::find(1);

echo 'User: ', $user->name, '<br>';

foreach ($user->photos as $photo) {

echo 'Photo: ', $photo->path . '<br>';

}

});

Route::get('post/photos', function () {

$post = Post::find(11);

echo 'Post: ', $post->title . '<br>';

foreach ($post->photos as $photo) {

echo 'Photo: ', $photo->path . '<br>';

}

});

Step 1. Find and retrieve valid user or post.

Step 2. Use the method containing the polymorphic call to photos to pull the correct photos.

What used to be placing a type column with a ‘U’ or ‘P’ is now considered a polymorphic relationship.

## Polymorphic relation the inverse

Look up the parent of a photo.

Code changes:

Photo.php:

public function imageable()

{

return $this->morphTo();

}

morphTo() – Returns the parent data for this child (photo).

Route.php

Route::get('photo/{id}', function ($id) {

$photo = Photo::findOrFail($id);

return $photo->imageable;

});

## Polymorphic relation many to many part 1

Create migrations and models for Video, Tag and Taggable.

Highlights from Migrations:

Videos:

public function up()

{

Schema::create('videos', function (Blueprint $table) {

$table->increments('id');

$table->string('name');

$table->timestamps();

});

}

Tags:

public function up()

{

Schema::create('tags', function (Blueprint $table) {

$table->increments('id');

$table->string('name');

$table->timestamps();

});

}

Taggable:

public function up()

{

Schema::create('taggables', function (Blueprint $table) {

$table->increments('id');

$table->integer('tag\_id');

$table->integer('taggable\_id');

$table->string('taggable\_type');

$table->timestamps();

});

}

Tag\_id – The id column value in the Tag model

Taggable\_Id– The id column value in the source model (here it’s Post or Video)

Taggable\_type – Name of the sourcing model

## Polymorphic relation many to many part 2

Preparing the models to retrieve data

Modify Post and Video models:

public function tags()

{

return $this->morphToMany('App\Tag', 'taggable');

}

morphToMany() – Define a polymorphic many-to-many relationship.

'App\Tag' – Related model.

'taggable' – Root to identify the \_id and \_type columns.

Modify Tag model:

public function posts()

{

return $this->morphedByMany('App\Post', 'Taggable');

}

public function videos()

{

return $this->morphedByMany('App\Video', 'Taggable');

}

morphedByMany() - Define a polymorphic, inverse many-to-many relationship.

'App\Post' – Related model.

'Taggable' – Root to identify the \_id and \_type columns.

## Polymorphic relation many to many – retrieving

Finally we migrated the migrations.

Created the following Routes:

List tags ties to a specific post:

Route::get('/post/tag', function () {

$post = Post::find(11);

echo 'Post - id: ', $post->id, ' title: ', $post->title, '<br>';

foreach ($post->tags as $tag) {

echo 'Tags: ', $tag->name, '<br>';

}

});

List tags ties to a specific video:

Route::get('/video/tag', function () {

$video = Video::find(1);

echo 'Video: ', $video->title . '<br>';

echo 'Video Tags: ', $video->tags, '<br>';

foreach ($video->tags as $tag) {

echo 'Tags:', $tag->name, '<br>';

}

});

## Polymorphic relation many to many - retrieving owner

Route to retrieve all the posts and videos associated with a particular tag.

Note the use of two for loops, one for posts and one for videos.

Route::get('/tag/all', function () {

$tag = Tag::find(3);

echo '<h1>Tag - id: ', $tag->id, ' Name: ', $tag->name, '</h1><br>';

foreach ($tag->posts as $post) {

echo '<h2>Post - id: ', $post->id, ' Name: ', $post->title, '</h2><br>';

}

foreach ($tag->videos as $video) {

echo '<h2>Video - id: ', $video->id, ' Name: ', $video->name, '</h2><br>';

}

});

For additional information on this whole unit see: <https://laravel.com/docs/5.2/eloquent-relationships>

# Section 12 - Laravel Fundamentals - Database - Tinker

## Section Intro with Edwin Diaz

Command line utility to interact with database (create tables, etc.)

## Creating data with tinker

To enter Tinker use the following command;

php artisan tinker

You can create records with;

$post = App\Post::create(

['title'=>'PHP Post From Tinker', 'content'=>'PHP Content from Tinker']

);

Confirmation message;

=> App\Post {#712

title: "PHP Post From Tinker",

content: "PHP Content from Tinker",

updated\_at: "2019-06-11 12:49:48",

created\_at: "2019-06-11 12:49:48",

id: 21,

}

You can redisplay what has been created by typing;

$post

Redisplayed record;

=> App\Post {#712

title: "PHP Post From Tinker",

content: "PHP Content from Tinker",

updated\_at: "2019-06-11 12:49:48",

created\_at: "2019-06-11 12:49:48",

id: 21,

}

Create a new blank record that has not been applied to the database;

$post = new App\Post

Confirmation the object (a blank record) has been created but not applied

App\Post {#707}

Set the values of the title and content columns. Tinker will echo the results of each statement to prove the change has been applied;

$post->title = "New Title From This Object"

$post->content = "Yeah baby I\'m coding and doing awesome"

Display the settings for the record;

$post

Results;

=> App\Post {#707

title: "New Title From This Object",

content: "Yeah baby I\'m coding and doing awesome",

}

To save the updated record;

$post->save()

Confirmation of the save;

=> true

Display the saved record

$post

Result;

=> App\Post {#707

title: "New Title From This Object",

content: "Yeah baby I\'m coding and doing awesome",

updated\_at: "2019-06-11 13:31:44",

created\_at: "2019-06-11 13:31:44",

id: 22,

}

You can issue SQL in tinker with the following;

$posts = DB::select('select \* from posts where id = ?', [21])

Result

=> [

{#720

+"id": 21,

+"title": "PHP Post From Tinker",

+"content": "PHP Content from Tinker",

+"created\_at": "2019-06-11 12:49:48",

+"updated\_at": "2019-06-11 12:49:48",

+"is\_admin": 0,

+"deleted\_at": null,

},

]

You can dump an entire table and store it in a variable with the following;

$posts = DB::select('select \* from posts')

Results;

=> [

{#703

+"id": 11,

+"title": "Title 1",

+"content": "Content For Title 1",

+"created\_at": "2019-05-28 19:58:51",

+"updated\_at": "2019-05-28 19:58:51",

+"is\_admin": 0,

+"deleted\_at": null,

},

***{Displays all records from posts}***

]

You can reprint the listing with;

$posts

Result;

{reprints the result of the select (listed above)}

## Finding record and using constraints in tinker

Finding a record or records using Tinker an models using find() method;

$post = App\Post::find(20)

Another way of doing the find using where() method. Works on any column.

$post = App\Post::where('id', 21)->first()

$post = App\Post::where('id', 21)->get()

You need the first() or get() or some other method that will actually retrieve record or only thing you’ve accomplished is to momentarily set the where condition.

There is a specific method to search by id, it’s called whereId(). See below;

$post = App\Post::whereId(11)->get()

## Updating and deleting with tinker

Going to find (retrieve), update, save, soft delete and permanently delete a record

Find:

$post = App\Post::find(21)

Result:

App\Post {#726

id: 21,

title: "PHP Post From Tinker",

content: "PHP Content from Tinker",

created\_at: "2019-06-11 12:49:48",

updated\_at: "2019-06-11 12:49:48",

is\_admin: 0,

deleted\_at: null,

}

Update:

$post->title = "updated record with id 3"

$post->content = "updated record content with id 3"

Save:

$post->save()

Result:

=> true

Soft Delete (because soft delete was enabled in the model):

$post->delete()

Result:

=> true

Retrieve those that have been soft deleted:

>>> $post = App\Post::onlyTrashed()

Permanently delete those that have been soft deleted:

>>> $post->forceDelete()

Result (number of records deleted):

=> 1

## Playing around with relations in tinker

Retrieve a user:

$user = App\User::find(3)

Result:

=> App\User {#729

id: 3,

name: "Jane",

email: "Jane.Doe@bsrresearch.com",

created\_at: "2019-05-31 19:00:00",

updated\_at: "2019-05-31 19:00:00",

country\_id: 6,

}

Bringing up the roles link to this user:

$user->roles

=> Illuminate\Database\Eloquent\Collection {#722

all: [

App\Role {#731

id: 6,

name: "No Access",

created\_at: "2019-05-31 12:00:00",

updated\_at: "2019-05-31 12:00:00",

pivot: Illuminate\Database\Eloquent\Relations\Pivot {#730

user\_id: 3,

role\_id: 6,

},

},

App\Role {#732

id: 5,

name: "security",

created\_at: "2019-05-31 12:00:00",

updated\_at: "2019-05-31 12:00:00",

pivot: Illuminate\Database\Eloquent\Relations\Pivot {#726

user\_id: 3,

role\_id: 5,

},

},

App\Role {#733

id: 4,

name: "editor",

created\_at: "2019-05-31 12:00:00",

updated\_at: "2019-05-31 12:00:00",

pivot: Illuminate\Database\Eloquent\Relations\Pivot {#727

user\_id: 3,

role\_id: 4,

},

},

App\Role {#734

id: 3,

name: "author",

created\_at: "2019-05-31 12:00:00",

updated\_at: "2019-05-31 12:00:00",

pivot: Illuminate\Database\Eloquent\Relations\Pivot {#728

user\_id: 3,

role\_id: 3,

},

},

],

}

Reverse it and look up the users linked to role:

Retrieve the role:

$role = App\Role::find(1)

Result:

=> App\Role {#725

id: 1,

name: "administrator",

created\_at: "2019-05-31 12:00:00",

updated\_at: "2019-05-31 12:00:00",

}

Retrieve the users link to the role

$role->users

Result:

=> Illuminate\Database\Eloquent\Collection {#721

all: [

App\User {#723

id: 1,

name: "Barry",

email: "barry.rayfield@gmail.com",

created\_at: "2019-05-29 22:30:00",

updated\_at: "2019-05-29 22:30:00",

country\_id: 1,

pivot: Illuminate\Database\Eloquent\Relations\Pivot {#724

role\_id: 1,

user\_id: 1,

},

},

],

}

For this to work the method belongsToMany() has to be properly implemented.

# Section 13: Database - Eloquent One to One Relationship CRUD

## Section intro with Edwin Diaz

—N/A—

## Creating a new Laravel installation / Setup

Create new directory with the Laravel core under tutorial called one2one for this set of lessons through the use of compose to create a new install of Laravel.

Composer create-project laravel/laravel ***one2one*** 5.2.31 --prefer-dist

***one2one*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Database configuration and migration

Edited .env file for your installation.

Created empty database one2one.

Created model and migration for Address table

php artisan make:model Address –m

## Setting up Relations

Added user\_id to Addresses and make it unsigned and nullable.

Added the following code to the User model;

public function address()

{

return $this->hasOne('App\Address');

}

Setting up the relation between user and address

## Creating data for user

Created user #1

Created the route;

Route::get('/insert', function () {

$user = User::findOrFail(1);

$address = new Address(

[

'street1'=>'79-30 256th Street',

'street2'=>'',

'city'=>'Floral Park',

'state'=>'NY',

'zip'=>'11004'

]

);

$user->address()->save($address);

});

Added $fillable to Addess.php to overcome a MassAssignmentException error;

protected $fillable = [

'user\_id', 'street1', 'street2', 'city', 'state', 'zip'];

## Updating Data

Added logic to Route.php to update the address.

Retrieval is done via a known user id of 1.

Route::get('/update', function () {

$address = Address::whereUserId(1)->first();

$address->street1 = '4353 Update Ave';

$address->save();

});

## Reading and deleting data

Added a read route:

Route::get('/read', function () {

$user = User::findOrFail(1);

echo $user->address->street1, '<br>';

if (!empty($user->address->street2)) {

echo $user->address->street2, '<br>';

}

echo $user->address->city, ', ';

echo $user->address->state, ' ';

echo $user->address->zip, '<br>';

});

Added a delete route () as in deleting the address and not the user):

Route::get('/delete', function () {

$user = User::findOrFail(1); //if you add delete() here you delete

// the user

$user->address()->delete(); //putting delete() here only deletes

// the address (all the addresses

// tied to this id in the address

// table

});

For more Information see: <https://laravel.com/docs/5.2/eloquent-relationships>

# Section 14: Database - Eloquent One to Many Relationship CRUD

## Creating a new Laravel installation / Setup

Create new directory with the Laravel core under tutorial called one2many for this set of lessons through the use of compose to create a new install of Laravel.

Composer create-project laravel/laravel ***one2many*** 5.2.31 --prefer-dist

***one2many*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Database migrations

Edited .env file for your installation.

Created empty database one2many.

Created model and migration for Address table

php artisan make:model Post –m

Applied migrations.

## Relationships and mass assignment

Modified User.php to request all posts for a particular user

public function posts()

{

return $this->hasMany('App\Post');

}

Modified Post.php to make fields mass assignable

protected $fillable = [

'title',

'body'

];

## Inserting Data

Create the route for inserting records in routes.php

Route::get('/create', function () {

$user = User::findOrFail(1);

// $post = new Post(

// [

// 'title'=>'My First Post',

// 'body'=>'I love Laravel, with Edviw Diaz'

// ]

// );

// $user->posts()->save($post);

$user->posts()->save(new Post(

['title'=>'My Second Post', 'body'=>'I love Laravel, with Edviw Diaz']

));

});

The above code shows two different forms of creating a post, as in ‘My First Post’ and ‘My Second Post’.

## Reading Data

Created route to read a user and the multiple posts linked to that user

Route::get('/read', function () {

$user = User::findOrFail(1);

//return $user->posts;

// dd($user->posts);

foreach ($user->posts as $post) {

echo 'Post: ' . $post->id . ' ' . $post->title . '<br>';

}

});

The function dd() is a Laravel variant on ver\_dump().

## Updating Data

Created route to update a specific post under a specific user

Route::get('/update', function () {

$user = User::find(1);

$user->posts()->whereId(1)->update(

[

'title'=>'I Love Laravel',

'body'=>'This is awesome, thank you Edwin'

]

);

});

## Deleting Data

Created route to delete a specific post under a specific user

Route::get('/delete', function () {

$user = User::find(1);

$user->posts()->whereId(1)->delete();

});

If you delete “whereId(1)->” from the route you end up deleting all the posts under a specific user.

For more information see: <https://laravel.com/docs/5.2/eloquent-relationships>

# Section 15: Database - Eloquent Many to Many Relationship CRUD

## Creating a new Laravel installation / Setup

Create new directory with the Laravel core under tutorial called many2many for this set of lessons through the use of compose to create a new install of Laravel.

Composer create-project laravel/laravel ***many2many*** 5.2.31 --prefer-dist

***many2many*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Database migrations

Edited .env file for your installation.

Created empty database many2many.

Created model and migration for Roles table

php artisan make:model Role –m

Created migration for role\_users table

php artisan make:migration create\_role\_user\_table --create=role\_user

Applied migrations.

## Relationships and mass assignment

Set up many to many relationship in Users

public function roles()

{

return $this->belongsToMany('App\Role');

}

Set up mass assignment in Role

protected $fillable = ['name'];

## Inserting Data

Manually create users.

Create route to create a role and the linkage to a user.

Route::get('/create', function () {

$user = User::find(1);

$role = new Role(['name'=>'Administrator']);

$user->roles()->save($role);

});

Optionally, the last 2 lines can be combined to form;

$user->roles()->save(new Role(['name'=>'Administrator']));

## Reading Data

Read from the roles table via the user table;

Route::get('/read', function () {

$user = User::findOrFail(1);

foreach ($user->roles as $role) {

//dd($role); //var dumper if you want to see

// the entire object

echo $role->name, '<br>';

}

});

## Updating Data

Retrieve the role via the user, update it and save it.

Route::get('/update', function () {

$user = User::findOrFail(1);

if ($user->has('roles')) {

foreach ($user->roles as $role) {

if ($role->name == 'Administrator') { //if the role's name is

// Administrator

$role->name = 'administrator'; //change the name to

// administrator

$role->save(); //and save it

}

}

}

});

## Deleting Data

Retrieve user and roles linked to user. Delete role whose id is 4.

Route::get('/delete', function () {

$user = User::findOrFail(1);

//$user->roles()->delete(); //delete all roles for this user

foreach ($user->roles as $role) {

//dd($role); //dump what role looks like

$role->whereId(4)->delete();

}

});

## Attaching, detaching and syncing

Link a user to an existing role in the role\_user table.

Route::get('/attach', function () {

$user = User::findOrFail(1);

$user->roles()->attach(2);

});

Remove a link from a user to an existing role in the role\_user table.

Route::get('/detach', function () {

$user = User::findOrFail(1);

$user->roles()->detach(2);

});

Remove all links from a user to an existing roles in the role\_user table.

Route::get('/detachall', function () {

$user = User::findOrFail(1);

$user->roles()->detach();

});

Create links from a user to existing roles in the role\_user table. In this case [2, 3, 5]. If any links exist outside of [2, 3, 5] for the user delete them.

Route::get('/sync', function () {

$user = User::findOrFail(1);

$user->roles()->sync([2,3,5]);

});

Note: There is no protection to prevent creation of a link to a role that does not exist. Need to check that first or referential integrity, etc.

For more information see: <https://laravel.com/docs/5.2/eloquent-relationships>

# Section 16: Database – Eloquent Polymorphic Relationship CRUD

## Creating a new Laravel installation / Setup

Create new directory with the Laravel core under tutorial called polymorphic for this set of lessons through the use of compose to create a new install of Laravel.

Composer create-project laravel/laravel ***polymorphic*** 5.2.31 --prefer-dist

***polymorphic*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Database migration

Edited .env file for your installation.

Created empty database polymorphic.

Created model and migration for Staff table

php artisan make:model Staff -m

Created model and migration for Products table

php artisan make:model Product -m

Created model and migration for Photos table

php artisan make:model Photo –m

Created migration for role\_users table

php artisan make:migration create\_role\_user\_table --create=role\_user

Applied migrations.

## Relationships and mass assignment

For the Photo model add the mass assignment ($fillable) and create the relationship (imageable())

protected $fillable = ['path'];

public function imageable()

{

return $this->morphTo();

}

For the Photo model add the mass assignment ($fillable) and create the relationship (photos())

protected $fillable = ['name'];

public function photos()

{

return $this->morphMany('App\Photo', 'imageable');

}

For the Photo model add the mass assignment ($fillable) and create the relationship (photos())

protected $fillable = ['name'];

public function photos()

{

return $this->morphMany('App\Photo', 'imageable');

}

## Inserting Data

Import classes

use App\Staff;

use App\Product;

use App\Photo;

Created routes to insert data into photos table

Add photos for staff

Route::get('/createforstaff', function () {

$staff = Staff::find(1);

$staff->photos()->create(['path'=>'staff photo 1.jpg']);

$staff->photos()->create(['path'=>'staff photo 2.jpg']);

$staff->photos()->create(['path'=>'staff photo 3.jpg']);

});

Add photos for products

Route::get('/createforproduct', function () {

$product = Product::find(1);

$product->photos()->create(['path'=>'product photo 1.jpg']);

$product->photos()->create(['path'=>'product photo 2.jpg']);

$product->photos()->create(['path'=>'product photo 3.jpg']);

});

## Reading Data

Route to read the staff data

Route::get('readforstaff', function () {

$staff = Staff::findOrFail(1);

echo 'Staff Name: ', $staff->name, '<br>';

foreach ($staff->photos as $photo) {

echo 'Path:', $photo->path, '<br>';

}

});

Route to read the product data

Route::get('readforproduct', function () {

$product = Product::findOrFail(1);

echo 'Product Name: ', $product->name, '<br>';

foreach ($product->photos as $photo) {

echo 'Path: ', $photo->path, '<br>';

}

});

## Updating Data

Updating a staff photo:

Route::get('/updateforstaff', function () {

$staff = Staff::findOrFail(1);

echo 'Staff Name: ', $staff->name, '<br>';

$photo = $staff->photos()->whereId(4)->first();

$photo->path = "Update Staff Photo.jpg";

$photo->save();

echo 'Updated<br>';

});

Updating a product photo:

Route::get('/updateforproduct', function () {

$product = Product::findOrFail(1);

echo 'Product Name: ', $product->name, '<br>';

$photo = $product->photos()->whereId(7)->first();

$photo->path = "Update Product Photo.jpg";

$photo->save();

echo 'Updated<br>';

});

## Deleting Data

Delete a photo for a staff member:

Route::get('/deleteforstaff', function () {

$staff = Staff::findOrFail(1);

echo 'Staff Name: ', $staff->name, '<br>';

$photo = $staff->photos()->whereId(4)->delete();

echo "Deleted Staff Photo<br>";

});

Delete a photo for a product:

Route::get('/deleteforproduct', function () {

$product = Product::findOrFail(1);

echo 'Product Name: ', $product->name, '<br>';

$photo = $product->photos()->whereId(7)->delete();

echo "Deleted Product Photo<br>";

});

Interesting link " How To Design Web and Mobile Apps in 7 Simple Steps”: <https://www.youtube.com/watch?v=HPjwv056FbA&feature=youtu.be&t=5m13s>

## Extra techniques

Assign an existing photo to a Staff member;

Route::get('/assign', function () {

$staff = Staff::findOrFail(1);

echo 'Staff Name: ', $staff->name, '<br>';

$photo = Photo::findOrFail(10);

echo 'Photo path:', $photo->path, '<br>';

$staff->photos()->save($photo);

echo 'Stafff member updated.', '<br>';

});

Un-assign a photo from a staff member but do not delete the photo:

Route::get('/un-assign', function () {

$staff = Staff::findOrFail(1);

echo 'Staff Name: ', $staff->name, '<br>';

$staff->photos()->whereId(6)->

update(['imageable\_id'=>'','imageable\_type'=>'']);

echo 'Staff photo unassigned.', '<br>';

});

Remember attach(), detach() and sync() do the same thing.

For more information see: <https://laravel.com/docs/5.2/eloquent-relationships>

# Section 17: Database – Eloquent Polymorphic Many to Many Relationships

## Creating a new Laravel installation / Setup

Create new directory with the Laravel core under tutorial called polymorphicmany2many for this set of lessons through the use of compose to create a new install of Laravel.

Composer create-project laravel/laravel ***polymorphicmany2many*** 5.2.31 --prefer-dist

***polymorphicmany2many*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Database migrations

Edited .env file for your installation.

Created empty database polymorphicmany2many.

Created model and migration for Posts table

php artisan make:model Post -m

Created model and migration for Videos table

php artisan make:model Video -m

Created model and migration for Tags table

php artisan make:model Tag -m

Created migration for Taggables table

php artisan make:model Taggable -m

Applied migrations.

## Relationships and mass assignment

Added the method tags() to Posts and Videos to create the relationships from Posts and Videos to Tags.

public function tags()

{

return $this->morphToMany('App\Tag', 'taggable');

}

Added $fillable to Posts, Videos and Tags to take care of mass assignment.

protected $fillable = ['name'];

## Inserting Data

Created route to seed tags, posts and videos tables. Added statements to create links from a post to a tag and a video to a tag.

Route::get('/create', function () {

$tag1 = Tag::create(['name'=>'Tag 1']); // create tags

$tag2 = Tag::create(['name'=>'Tag 2']);

$tag3 = Tag::create(['name'=>'Tag 3']);

$tag4 = Tag::create(['name'=>'Tag 4']);

$post = Post::create(['name'=>'Post 1']); // create posts

$post2 = Post::create(['name'=>'Post 2']);

$post3 = Post::create(['name'=>'Post 3']);

$post4 = Post::create(['name'=>'Post 4']);

$tag1 = Tag::find(1);

$post->tags()->save($tag1); // create link from 1st

// post to 1st tag

$video = Video::create(['name'=>'Video 1.mov']); // create videos

$video2 = Video::create(['name'=>'Video 2.mov']);

$video3 = Video::create(['name'=>'Video 3.mov']);

$video4 = Video::create(['name'=>'Video 4.mov']);

$tag2 = Tag::find(2);

$video->tags()->save($tag2); // create link from 1st

// video to 2nd tag

});

## Reading and updating data

Created route with various ways of doing updates:

Route::get('/update', function () {

$post = Post::findOrFail(17);

echo 'Post: ', $post->name, '<br>';

foreach ($post->tags as $tag) {

echo $tag->whereName('Tag 1 - Updated')->

update(['name'=>'Tag 1 - Updated']);

}

echo '<br><br>';

$post = Post::findOrFail(17);

echo 'Post: ', $post->name, '<br>';

$tag = Tag::find(2);

$tag3 = Tag::find(3);

echo $post->tags()->save($tag), '<br>';

echo $post->tags()->attach($tag3), '<br>';

$post->tags()->sync([1, 2]);

});

## Deleting Data

Created route to delete tags, not the records in the taggable table

Route::get('/delete', function () {

$post = Post::findOrFail(17);

foreach ($post->tags as $tag) {

$tag->whereId(3)->delete();

}

});

To delete the records from the taggable table see detach().

For more information see: <https://laravel.com/docs/5.2/eloquent-relationships>

# Section 18: Forms and Validation

## Section Intro with Edwin Diaz

Welcome you will be doing forms and validation.

## Setting up migration and relations

Original intent was to reuse cms. But since it requires wiping the data I created cms2 as a project and laravel\_cms2 as the database.

We are using the migrations and models from cms/laravel\_cms.

Changes so far:

In the posts migration the column user\_id is now included instead of commented out.

$table->integer('user\_Id')->unsigned();

In user model added the following methods;

public function post()

{

return $this->hasOne('App\Post');

}

public function posts()

{

return $this->hasMany('App\Post');

}

public function roles()

{

return $this->belongsToMany('App\Role');

//return $this->belongsToMany('App\Role')->withPivot();

//return $this->belongsToMany('App\Role')->withPivot('created\_at');

}

public function photos()

{

return $this->morphMany('App\Photo', 'imageable');

}

## Setting up views and routes

Going to use the existing routes provided by the postcontroller:

+--------+-----------+--------------------+---------------+------------------------+------------+

| Domain | Method | URI | Name | Action | Middleware |

+--------+-----------+--------------------+---------------+------------------------+------------+

| | GET|HEAD | / | | Closure | web |

| | POST | posts | posts.store | App\Http\Controllers\ | web |

| | | | | PostController@store| |

| | GET|HEAD | posts | posts.index | App\Http\Controllers\ | web |

| | | | | PostController@index| |

| | GET|HEAD | posts/create | posts.create | App\Http\Controllers\ | web |

| | | | | PostController@create| |

| | GET|HEAD | posts/{posts} | posts.show | App\Http\Controllers\ | web |

| | | | | PostController@show| |

| | PUT|PATCH | posts/{posts} | posts.update | App\Http\Controllers\ | web |

| | | | | PostController@update| |

| | DELETE | posts/{posts} | posts.destroy | App\Http\Controllers\ | web |

| | | | | PostController@destroy| |

| | GET|HEAD | posts/{posts}/edit | posts.edit | App\Http\Controllers\ | web |

| | | | | PostController@edit| |

+--------+-----------+--------------------+---------------+------------------------+------------+

Created four empty view files:

{application root}\resources\posts\create.blade.php

{application root}\resources\posts\edit.blade.php

{application root}\resources\posts\index.blade.php

{application root}\resources\posts\show.blade.php

## 4-Setting the markup for create

Created a master layout in the directory

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<div class="container">

@yield('content')

</div>

@yield('footer')

</body>

</html>

For more information on master layout see lesson 36.

Modify the view create.blade.php with the following;

@extends(‘layouts.app')

@section('content')

<form method="post" action="/posts">

<input type="text" name="title" placeholder="Enter Title">

<input type="submit" name="submit">

</form>

@yield('footer')

@extends(layouts.app') Identifies the master layout to use. In this case the layout name is app.blade.php and it is located in the subdirectory layouts.

@section('content') Identifies content that will be substituted in the area labeled @yield(‘content’) in the master layout.

@yield('footer') Not defined yet.

## Setting up the controller and view

Due to my unique setup the URL in the form action has to change. Action is now set to:

“http://tutorial.tst/cms2/public/posts”

This is something to take into consideration for the rest of the course.

In the form the line {{ csrf\_field() }} has to be added after the title input field. This fixes a TokenMisatchException error that can be thrown.

For more information on cross-site request forgery protection or CSRF see:

<https://laravel.com/docs/5.8/csrf>

Changes made to postscontroller.php

Method create:

public function create()

{

return view('posts.create');

}

Calls the view (web page) create.blade.php located in the resources\views\posts subdirectory

Method store:

public function store(Request $request)

{

// return $request->all();

// $value .= '<br>';

// $value .= 'title: ' . $request->get('title');

// return $value;

// return '<br>title: ' . $request->title . '<br>';

return '<br>title: ' . $request->get('title') . '<br>';

}

$request->all() Returns the contents of Request as an array.

$request->get('title') Returns the value of title or $\_POST[‘title’].

$request->title Same as above.

## Persisting data to database

Modifying the store() method in postscontroller.php to store the data in the posts table.

Modify postscontroller with the following:

Add use App\Post; at the top if it not already there.

Modify the store() method with the following:

public function store(Request $request)

{

Post::create($request->all());

-- OR --

$post = new Post();

$post->title = $request->title;

$post->save();

}

## Reading data

Modify store() method in postcontroller.php to redirect to the index() method in postcontroller.php after saving to the posts table is complete.

public function store(Request $request)

{

Post::create($request->all());

return redirect('/posts');

}

Modify index() method in postcontroller.php to read the data from the posts table and send it to the index.blade.php view.

public function index()

{

$posts = Post::all();

return view('posts.index', compact('posts'));

}

compact('posts') - Creates an array containing variables and their values. For each of these, compact() looks for a variable with that name in the current symbol table and adds it to the output array such that the variable name becomes the key and the contents of the variable become the value for that key. In short, it does the opposite of extract().

This is needed because $posts is not an array and need to be converted to one.

Modify the view index.blade.php

@extends('layouts.app') {{-- extends master layout layouts\app.blade.php --}}

@section('content') {{-- replace content section with this --}}

<ul> {{-- create unordered list --}}

@foreach ($posts as $post) {{-- loop through each post --}}

<li>{{$post->title}}</li> {{-- turn each post’s title into a list entry --}}

@endforeach

</ul>

@endsection

## Showing individual posts and updating

Modified the show() method in postscontroller.php method.

public function show($id)

{

$post = Post::findOrFail($id);

return view('posts.show', compact('post'));

}

As opposed to the index() method the show() displays only one record not all.

Modified the view index.blade.php

<ul>

@foreach ($posts as $post)

<li>

<a href="{{route('posts.show', $post->id)}}">

{{$post->title}}

</a>

</li>

@endforeach

</ul>

Each item in the list is now a link to the show() method passing the id of a post.

Modifiiied the view edit.blade.php

@extends('layouts.app')

@section('content')

<h1>Edit Post</h1>

{{-- <form method="post" action="/posts"> --}}

{{-- <form method="post" action="{‌{ route('posts.store') }}"> --}}

<form method="post" action="http://tutorial.tst/cms2/public/posts">

<input type="text" name="title" placeholder="Enter Title">

{{ csrf\_field() }}

<input type="submit" name="submit">

</form>

@endsection

Initial setup of the edit.blade.php. It’s copied from create.blade.php. Needs further editing.

Modified the edit() method in postscontroller.php method.

public function edit($id)

{

$post = Post::findOrFail($id);

return view('posts.edit', compact('post'));

}

Find a specific post and pass that data to the view edit.blade.php.

## Preparing the edit view and setting up

Finished the view edit.blade.php

<h1>Edit Post</h1>

<form method="post" action="/cms2/public/posts/{{$post->id}}">

{{ csrf\_field() }}

<input type="hidden" name="\_method" value="PUT">

<input type="text" name="title" placeholder="Enter Title" value="{{$post->title}}">

<input type="submit" name="submit">

</form>

As I do not have the URI rooted all the way down to public my action=”” is longer than expected. It needs more than /posts/{{$post->id}}

## 10- Editing records and redirecting

Modified the update method of postscontroller.php

public function update(Request $request, $id)

{

$post = Post::findOrFail($id);

$post->update($request->all());

return redirect('/posts');

}

Modified the view show.blade.php

@section('content')

<h1><a href="{{route('posts.edit', $post->id)}}">{{$post->title}}</a></h1>

@endsection

The post is now a link to the edit page.

## 11- Deleting

Modified destroy() method in postscontroller.php:

public function destroy($id)

{

$post = Post::findOrFail(15);

$post->delete();

return redirect('/posts');

}

Find it. Delete it. Go to index page.

Modified view edit.blade.php:

<form method="post" action="/cms2/public/posts/{{$post->id}}">

{{ csrf\_field() }}

<input type="hidden" name="\_method" value="DELETE">

<input type="submit" name="Delete" value="Delete">

</form>

Added the Delete button in its own form. Requires adding {{ csrf\_field() }}. Needs a hidden field named \_method with a value of “DELETE”.

# Section 19: Forms - Package and Validation

## Section intro with Edwin Diaz

Use a package for forms (generation of HTML) and do validation.

## UPDATE – IMPORTANT

Warning about compatibility issue with LaravelCollectivte\HTML (LCH) and Laravel (L)

If you are using L 5.4 or higher do not use LCH. It only works with L5.3.+ and lower

For those using L 5.2.+ at this point the command to install it is:

composer require 'laravelcollective/html:5.2'

Possible syntax error, could also try:

composer require laravelcollective/html 5.2

Another possibility:

composer require "laravelcollective/html":"^5.2.0"

## Installing package and testing

Package: Laravel illuminate/html

Automated install: composer require laravelcollective/html

Manual install:

Edit {application root}\composer.json.

Go to the section labeled "require".

Place a comma “,” after the last entry in that section and add a new line.

Type in “laravelcollective/html”: “5.2.\*”

Save and open a terminal window in the cms2 directory.

At the prompt enter “composer update” and hit enter.

Edit {application root}\config\app.php.

Go to the 'providers' section.

Add the following: Collective\Html\HtmlServiceProvider::class,

Go to the ‘alias’ section and add the following:

'Form' => Collective\Html\FormFacade::class,

'Html' => Collective\Html\HtmlFacade::class,

Test it.

Edit the view create.blade.php

Comment out the <form> line and change it to this new notation as shown below:

Original: {{-- <form method="post"

action="http://tutorial.tst/cms2/public/posts"> --}}

New: {!! Form::open() !!}

Save and test it by going to: <http://tutorial.tst/cms2/public/posts/create>

Inspect it. You should see:

<form method="POST" action="http://tutorial.tst/cms2/public/posts/create" ***accept-charset="UTF-8">***

If you see accept-charset=”UTF-8” it’s working.

## Modifying our create form with the form package

Modify the view create.blade.php to use the laravelcollective\html package

Original view:

@extends('layouts.app')

@section('content')

<h1>Edit Post</h1>

<form method="post" action="/cms2/public/posts">

<input type="text" name="title" placeholder="Enter Title">

{{ csrf\_field() }}

<input type="submit" name="submit">

</form>

@endsection

Modified version:

@section('content')

<h1>Create Post</h1>

{!! Form::open(['medthod'=>'POST', 'action'=>'PostsController@store']) !!}

<div class="form-group">

{!! Form::label('title', 'Title:') !!}

{!! Form::text('title', null, ['class'=>'form-control']) !!}

</div>

<div class=form-group>

{!! Form::submit('Create Post', ['class'=>'btn btn-primary']) !!}

</div>

{!! Form::close() !!}

@endsection

Form::label('title', 'Title:') generates <label for="title">Title:</label>

First parameter is the ‘for’ parameter. Second parameter is the actual text for the label. There is a third parameter (not shown) it is an array for the class, etc.

Form::text('title', null, ['class'=>'form-control']) generates <input class="form-control" name="title" type="text" id="title">

The first parameter is the name and id. Second parameter is the default value. The third parameter is an array for the class, etc.

Form::submit('Create Post', ['class'=>'btn btn-primary']) generates <input class="btn btn-primary" type="submit" value="Create Post">

The first parameter is the value for the button. Second parameter is an array for the class, etc.

laravelcollective\html will take care of issuing the ‘{{ csrf\_field() }}’ method for any form opened with the POST method.

For more Documentation on laravelcollective\html see: <https://github.com/LaravelCollective/docs/blob/5.2/html.md#labels>

## Modifying our edit and delete forms with the form package

Converted the view edit.blade.php to laravelcollective\html See result below:

@extends('layouts.app')

@section('content')

<h1>Edit Post</h1>

{!! Form::model($post, ['method'=>'PATCH',

'action'=>['PostsController@update', $post->id]]) !!}

<div class="form-group">

{!! Form::label('title', 'Title:') !!}

{!! Form::text('title', null, ['class'=>'form-control']) !!}

</div>

<div class=form-group>

{!! Form::submit('Update Post', ['class'=>'btn btn-primary']) !!}

</div>

{!! Form::close() !!}

{!! Form::open(['method'=>'DELETE',

'action'=>['PostsController@destroy', $post->id]]) !!}

<div class=form-group>

{!! Form::submit('Delete Post', ['class'=>'btn btn-danger']) !!}

</div>

{!! Form::close() !!}

@endsection

For the delete button you have to use PATCH and not PUT even though the route list saya you can use either. For more information see: <https://laracasts.com/series/laravel-5-fundamentals/episodes/13>

## Basic validation

Two ways to validate data, utilize the validate() method built into the PostsCotroller class’s parent class Controller or create a Request class.

This lesson covers the validate() method.

Forces modification to store() method in postscontroller.php

public function store(Request $request)

{

$this->validate($request, [

'title'=>'required',

'content'=>'required'

]);

Post::create($request->all());

return redirect('/posts');

}

validate() passes the data ($request) and an array of rules to the method.

If validate() fails you will be thrown back to the parent, be it create() or edit() in this case, and an array $errors will be loaded with a list of failed rules.

Displaying $error will be covered in the next lesson.

## Displaying errors

To access $errors you have to modify routes.php and create.blade.php

Changes to routes.php

Route::group(['middlewareGroups' => 'web'], function () {

Route::resource('/posts', 'PostsController');

});

Originally instructed to enclose Route::resource() within a Route::group() having middlewareGroups as middleware. Turns out there in an issue with $errors, Route::resource() and Route::group([‘middleware’…) in Laravel 5.2. For the moment you can comment out the Route::group() code or change middleware to middlewareGroups. For more information see: <https://www.udemy.com/php-with-laravel-for-beginners-become-a-master-in-laravel/learn/lecture/4911058#questions/7213824> and <https://stackoverflow.com/questions/34438463/laravel-5-2-errors-not-appearing-in-blade>

Changes to create.blade.php (added just above @endsection)

@if(count($errors)>0)

<div class="alert alert-danger">

<ul>

@foreach($errors->all() as $error)

<li>{{$error}}</li>

@endforeach

</ul>

</div>

@endif

If you want to use custom error messages see: <https://stackoverflow.com/questions/47918872/how-to-set-laravel-custom-validation-message-with-laravel-rules> and <https://stackoverflow.com/questions/49432025/laravel-5-5-validate-with-custom-messages>

## Advance validation

Create the Request class.

php artisan make:request CreatePostRequest

Created file: {application root}\Http\Requests\CreatePostRequest.php

Final form of CreatePostRequest.php is;

<?php

namespace App\Http\Requests;

use App\Http\Requests\Request;

class CreatePostRequest extends Request

{

/\*\*

\* Determine if the user is authorized to make this request.

\*

\* @return bool

\*/

public function authorize()

{

return true;

}

/\*\*

\* Get the validation rules that apply to the request.

\*

\* @return array

\*/

public function rules()

{

return [

'title'=>'required'

];

}

}

Authorize() method is forced to return true as this is an unrestricted “application”. There are no unauthorized users.

rules() method actually does the validation. The rules that were in the validate() method of postscontroller.php were relocated here.

Modify postscontroller.php

Add the following line at the top;

use App\Http\Requests\CreatePostRequest;

Modify the header of the store() method adding the CreatePostRequest data type:

public function store(CreatePostRequest $request)

Comment out or remove the call to the validate() method in the store() method.

The validation can now be used not only in postscontroller but in other classes that could use it.

## Creating snippets with PhpStorm

Laravel validation: <https://laravel.com/docs/5.2/validation>

# Section 20: Database - Some more model manipulation

## Section Intro with Edwin Diaz

Nothing to note

## Dates

See PHP documentation for PHP date objects and functions.

Laravel comes with Carbon which provides addition date methods and functions.

If you want to reinstall Carbon do the following;

Search for it.

composer search carbon

Result:

nesbot/carbon A simple API extension for DateTime.

Install:

composer require nesbot/carbon

For more information on Carbon objects and functions see: <https://carbon.nesbot.com/docs/>

## Accessors

Scenario: you have a route to return a user’s name but the name should be modified before it is returned or you don’t want the programmer to have direct access to the name.

The following is added to route.php

Route::get('/getname', function () {

$user = User::find(1);

echo 'User: ', $user->name;

});

Solution: Provide a method to manipulate the name before returning it.

You add the following to the model user.php

public function getNameAttribute($value)

{

return strToUpper($value);

}

The format of the name is dictated.

get - As this is an accessor

Name - column being accessed

Attribute - Laravel requirement

Laravel will look for get{column name}Attribute() method when it retrieves a column and if one is defined it will use it.

## Mutators

Scenario: you have a route to save a user’s name to the database but the name should be modified before it is saved or you don’t want the programmer to have direct access to the name.

The following is added to route.php

Route::get('/setname', function () {

$user = User::find(1);

$user->name = 'lower case';

$user->save();

});

Solution: Provide a method to manipulate the name before saving it.

You add the following to the model user.php

public function setNameAttribute($value)

{

$this->attributes['name'] = strToUpper($value);

}

The format of the name is dictated.

set - As this is an mutator

Name - column being accessed

Attribute - Laravel requirement

Laravel will look for set{column name}Attribute() method when it tries to save an object to the database and if one is defined it will use it.

## Query Scope

Limiting the number of and sorting the records returned from the database.

Changes to PostsController.php index() method;

public function index()

{

//$posts = Post::all(); // return all posts

//$posts = Post::latest()->get(); // return all in desc order by date

//$posts = Post::orderBy('id', 'desc')->get();

// return all in desc order by id

//$posts = Post::orderBy('id', 'asc')->get();

// return all in asc order by id

$posts = Post::latest(); // create a customized latest method in

// post.php

return view('posts.index', compact('posts'));

}

Changes to post.php;

public static function scopeLatest($query)

{

return $query->orderBy('id', 'asc')->get();

}

Laravel method header naming convention:

scope - Required by Laravel

Latest - Name of method used in PostsController index() method

but the first letter of the original method is now capitalized.

($query) - What data is being sent to the method.

Inside the method you set the sort and range of data to be returned and apply it to $query. In this case get all posts and order by id in ascending order.

For more information on accessors and mutators see:

<https://laravel.com/docs/5.7/eloquent-mutators#accessors-and-mutators>

<https://laravel.com/docs/5.4/eloquent-mutators#defining-a-mutator>

<https://laravel.com/docs/5.2/eloquent-mutators>

For more information on query scope see:

<https://laravel.com/docs/5.2/eloquent#query-scopes>

# Section 21: Forms - Uploading files

## Section intro with Edwin Diaz

Will be learning how to upload files.

## UPDATE – IMPORTANT – ABOUT UPLOADING

In section 19 you were warned about LaravelCollective/HTML, and that only you guys using 5.2, 5.3 or 5.4 version of Laravel could use it because it was not maintained anymore, but I also told you that I don't recommend using it at all.

You are going to see me using it to show you how upload files, so I'm creating this small article to show you how to create a regular form for uploads without using this package.

Example of a regular form for files

<form action="/uploadfile" method="post" enctype="multipart/form-data">

@csrf

<div class="form-group">

<input type="file"

class="form-control-file"

name="fileToUpload"

id="exampleInputFile">

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

## Modifying our create view for file input

Modified the view create.blade.php to show a choose file button after the title;

<div class="form-group">

{!! Form::file('file', ['class'=>'btn- btn-primary']) !!}

</div>

## Retrieving file data

When the user clicks on the “***Create Post***” button they are sent to the store() method of PostsController.php. From there you can produce a variety of information on the file as shown;

public function store(CreatePostRequest $request)

{

$file = $request->file('file');

echo '<br>';

echo 'original name:', $file->getClientOriginalName(), '<br>';

echo 'original Size:', $file->getClientSize(), '<br>';

}

When uploading files keep an eye on certain php.ini settings:

file\_uploads

post\_max\_size

upload\_max\_filesize

memory\_limit

max\_input\_time

upload\_tmp\_dir

max\_execution\_time

Along with a setting in httpd-fastcgi.conf:

FcgidMaxRequestLen

See notes from the Udemy course: PHP OOP - Object Oriented Programming For Beginners

## Persisting file data into the database

Creation of migration (should have used the photos table):

public function up()

{

Schema::table('posts', function (Blueprint $table) {

$table->string('path');

});

}

public function down()

{

Schema::table('posts', function (Blueprint $table) {

$table->dropColumn('path');

});

}

Create and/or drop a column in the posts table to hold the name of the photo.

Modify store() method in PostsController.php:

public function store(CreatePostRequest $request)

{

$input = $request->all();

if ($file = $request->file('file')) {

$name = $file->getClientOriginalName();

$file->move('images', $name);

$input['path'] = $name;

}

Post::create($input);

return redirect('/posts');

}

If the name of the photo exists (if the photo exists) reset the value of $file with the original filename.

$request->file('file') returns the temp file name php assigns to an uploaded picture. The last part of the if statement makes sure the file is moved and renamed back to its original name.

Modify Post.php:

protected $fillable = [

'title',

'content',

'path'

];

Add the column path to $fillable thereby making it mass assignable.

## Displaying images and using accessors to make it easy

Modify model post.php:

public $directory = 'images/';

. . .

. . .

. . .

public function getPathAttribute($value)

{

return $this->directory . $value;

}

Define $directory which will be used as a prefix for the value in the path column.

Created an accessor (getPathAttribute($value)) to reformat the value in the path column.

getPathAttribute($value) will add the directory name ($this->directory) to the value in the path column ($value).

Modify index.blade.php:

<ul>

@foreach ($posts as $post)

<div class="image-container">

<img height="100" src="{{ $post->path }}" alt="No Picture">

</div>

<li>

<a href="{{route('posts.show', $post->id)}}">{{$post->title}}</a>

</li>

@endforeach

</ul>

Added the <img> statement to display the image referred to in the path column.

Because of the accessor (getPathAttribute($value)) in the posts.php model you can just use {{ $post->path }} instead of having to tack on the directory name images/.

For more information see: <https://laravel.com/docs/5.2/requests#files>

# Section 22: Form - Login

## Section intro with Edwin Diaz

We are about to learn how Laravel handles login services.

## New Laravel Installation

Create new directory with the Laravel core under tutorial called laravellogin for this set of lessons through the use of compose to create a new install of Laravel.

Composer create-project laravel/laravel ***laravellogin*** 5.2.31 --prefer-dist

***laravellogin*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Database connection and migrations

Edited .env file for your installation.

Created empty database laravel\_login.

Using basic model and migration for users and password resets table.

php artisan migrate

Migration table created successfully.

Migrated: 2014\_10\_12\_000000\_create\_users\_table

Migrated: 2014\_10\_12\_100000\_create\_password\_resets\_table

## Creating the login system under 1 minute

Create a login front end under Laravel:

{Application root}>***php artisan make:auth***

Created View: {Application root}\resources/views/auth/login.blade.php

Created View: {Application root}\resources/views/auth/register.blade.php

Created View: {Application root}\resources/views/auth/passwords/email.blade.php

Created View: {Application root}\resources/views/auth/passwords/reset.blade.php

Created View: {Application root}\resources/views/auth/emails/password.blade.php

Created View: {Application root}\resources/views/layouts/app.blade.php

Created View: {Application root}\resources/views/home.blade.php

Created View: {Application root}\resources/views/welcome.blade.php

Installed HomeController.

Updated Routes File.

Authentication scaffolding generated successfully!

Under {Application root}\resources/views/,

{Application root}\resources/views/auth/,

{Application root}\resources/views/layouts/

{Application root}\app\http

{Application root}\app\http\Controllers

You will find views for welcome, login/register and password email/reset

Routes and controllers to run all this as well.

***You must run this first!!!*** It will overwrite existing files.

Now just have to hook it up to your application.

You have been left at the point where you can now register new users and their passwords are encrypted.

It would be better to run php artisan make:auth which will create the necessary files. And then you can run php artisan migrate. That would be the best order of action.

## Login creation overview

Take a tour through the various files that have been generated.

## Retrieving authenticated user data

Modification to routes.php

Route::get('/', function () {

return view('welcome');

});

Typical “/” route showing the Auth Welcome which checks to see if a user is authenticated behind the scenes.

Route::get('/', function () {

if (Auth::check()) {

return "The user is logged in.";

}

});

Alternate version of the “/” route showing an explicit check on a user to see if they are authorized.

Route::get('/', function () {

$username = “abcdef”;

$password = “nvonifhf8249yfhu”

if (Auth::attempt(['username'=>$username, 'password'=>$password])) {

return redirect()->intended('/admin');

}

});

Alternate version of the “/” route utilizing the attempt() method. The attempt() method accepts an array of key / value pairs as its first argument. The values in the array will be used to find the user in the user table for authorization.

Auth::logout();

Method used to force a logout of the current user.

Modifications to HomeCountroller.php

Original index() method:

public function index()

{

return view('home');

}

Modified index() method for passing user data to the home view:

public function index()

{

$user = Auth::user();

return view('home', compact('user'));

}

User data is retrieved with a call to the method Auth::user(). That data is then run through the compact function and passed to the view.

Modified home view home.blade.php to show user data:

@section('content')

<div class="container">

<div class="row">

<div class="col-md-10 col-md-offset-1">

<div class="panel panel-default">

<div class="panel-heading">Dashboard</div>

<div class="panel-body">

You are logged in! {{$user->name}}

</div>

</div>

</div>

</div>

</div>

@endsection

In the view it can now be used like any other data.

For more information see: <https://laravel.com/docs/5.2/authentication>

# Section 23: Middleware - Security / Protection

## Section intro

HTTP middleware provides a convenient mechanism for filtering HTTP requests entering your application.

Laravel comes with middleware to authenticate users (Auth), is effectively a filter to permit or deny authenticated users through, maintenance, authentication, CSRF protection, and more

Create new installation called middleware.

## Create a new laravel installation (challenge)

Create a new Laravel installation:

Composer create-project laravel/laravel middleware 5.2.31 --prefer-dist

Apply supplied migrations (user & password):

php artisan migrate

Create login front end:

php artisan make:auth

Middleware Review:

Middleware resides in {application root}\app\Http

Middleware is invoked by {application root}\app\Http\Kernel.php

Core, custom and user written middleware resides in:

{application root}\app\Http\Middleware

To create a skeleton for a middleware class use the following;

php artisan make:middleware RoleMiddleware

The file {application root}\app\Http\Middleware\RoleMiddleware.php was created.

## Registering a new middleware and using it

To place an app in maintenance mode (offline):

php artisan down

By default shows a message centered on the screen “Be Right Back”. It’s changeable.

To bring the application back on line.

php artisan up

In Kernel.php the section that lets you drop into maintenance mode is:

protected $middleware = [

\Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,

];

In the previous lesson we created the middleware RoleMiddleware. To register it with Laravel add it to the $routeMiddleware property in Kernel.php. See the modification in ***bold / italic / underline / in blue***.

protected $routeMiddleware = [

'auth' => \App\Http\Middleware\Authenticate::class,

'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

'can' => \Illuminate\Foundation\Http\Middleware\Authorize::class,

'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,

***'role' => \App\Http\Middleware\RoleMiddleware::class,***

];

Two ways to access the middleware

1. Via a class such as HomeController.php.

public function \_\_construct()

{

$this->middleware('auth');

}

1. Via routes.php. See example below.

Route::get('/admin/user/roles', ['middleware' => 'role', function () {

return 'Middleware role';

}]);

The return at the moment is meaningless as it does not appear anywhere.

‘role’ is a reference to an element in the array in the $routeMiddleware property of the Kernel class of Kernel.php which points to the RoleMiddleware class in RoleMiddleware.php.

RoleMiddleware Class (the handle() method is automatically invoked by the call):

public function handle($request, Closure $next)

{

return redirect('/');

}

In the handle() method the user is redirected to the home page.

## A more practical way to use middlewares 1 - roles, migration a…

Prep: Restore RoleMiddleware.php to its original state. Delete return redirect('/');

Scenario:

* Create roles table
* Assign roles to those in the users table.
* Create middleware for users with a role of admin.

Create a migration and model for the Roles table

php artisan make:model Role –m

Modify the users table migration to include a role\_Id.

Modify the roles table migration to include a name.

As there is almost no data run a refresh of the migrations.

php artisan migrate:refresh

Setup relation.

In the Roles model create the $fillable property to make the name mass updatable

In the Users model create the relationship with the role() method.

public function role()

{

return $this->belongsTo('App\Role');

}

## A more practical way to use middlewares 2 - Custom method

Created new middleware class IsAdmin. No editing yet.

Modified Kernel class. Added line below to the end of $routeMiddleware array property.

'IsAdmin' => \App\Http\Middleware\IsAdmin::class,

Modified User class to add an isAdmin() method testing the role and returning true or false.

public function isAdmin()

{

if ($this->role->name == 'administrator') {

return true;

}

return false;

}

Momentarily modified the home (‘/’) route to test the isAdmin() method of the User class.

Route::get('/', function () {

$user = Auth::user();

if ($user->isAdmin()) {

echo "This user is an administrator.";

}

// return view('welcome');

});

## A more practical way to use middlewares 3 - Custom method

Modified middleware IsAdmin:

public function handle($request, Closure $next)

{

$user = Auth::user();

if ($user->isAdmin()) {

return redirect()->intended('/admin');

}

return $next($request);

}

Modified model User:

public function isAdmin()

{

if ($this->role->name == 'administrator') {

return true;

}

return false;

}

Modified Routes:

Route::get('/', function () {

return view('welcome');

});

. . .

. . .

. . .

Route::get('/admin', 'AdminController@index');

Created new controller AdminController:

php artisan make:controller AdminController

Added the following methods:

public function \_\_construct()

{

$this->middleware('IsAdmin');

}

public function index()

{

return 'You are and administrstor because you are seeing this page';

}

## A more practical way to use middlewares 4 - Custom method

Testing

Bug 1: Had to add ‘use Illuminate\Support\Facades\Auth;’ to IsAdmin middleware.

Bug 2: Logic bug in handle method of IsAdmin middleware.

Change to:

if (!$user->isAdmin()) {

return redirect('/');

}

Original tried to implement the exact reverse. If you were not and admin you saw the admin prompt and if you were and admin you got stuck in an infinite loop.

For more information see: <https://laravel.com/docs/5.2/middleware>

# Section 24: Laravel Sessions

## Section intro with Edwin Diaz

Welcome to lecture on sessions.

## Setting and reading sessions

Modified the controller HomeController:

public function index(Request $request)

{

$request->session()->put(['edwin' => 'master instructor']);

session(['peter' => 'student']);

return $request->session()->all();

// return view('home');

}

Added Request $request to the header to gain access to the session methods

$request->session() is the Laravel Request class’ way of providing session methods.

session() is the regular php function for accessing session data.

Laravel Service Container: <https://laravel.com/docs/5.7/container>

Type Hinting: <https://phpenthusiast.com/object-oriented-php-tutorials/type-hinting>

<https://www.sitepoint.com/type-hinting-in-php/>

<https://stackoverflow.com/questions/38308985/why-is-type-hinting-necessary-in-php>

## Global session function deleting

To delete a single key in the session: $request->session()->forget('peter');

To delete all session keys: $request->session()->flush();

## Flashing data

Flashing data will only last for one request (“screen” or “refresh”).

The following will store the message under flash and display it.

$request->session()->flash('message', 'Posts has been created');

return $request->session()->get('message');

When you redisplay the screen or refresh etc. the key “message” is gone.

Additional flash methods;

$request->session()->reflash(); Will keep all of the flash data for an additional request.

$request->session()->keep([key1, key2, key3, keyn]); Will keep specified keys saved under flash for an additional request.

For more information see: <https://laravel.com/docs/5.2/session>

# Section 25: Laravel - Sending Email / Api

## Section intro with Edwin Diaz

Intro.

Using native php mail functions will send email to spam folder

To avoid this we will be using an API and Mailgun.

## New laravel installation

Create new directory with the Laravel core under tutorial called mail for this set of lessons through the use of compose to create a new install of Laravel.

Composer create-project laravel/laravel ***mail*** 5.2.31 --prefer-dist

***mail*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Overview on mailgun

@mailgun The Email Service For Developers

Link: <https://www.mailgun.com/>

Id: barry.rayfield@gmail.com

Password: CockSucker069

Other email services: <http://www.pfind.com/alternatives/mailgun>

Very good resource that provides in-depth info (some duplicates that are mentioned in the first link): <https://easysendy.com/blog/most-reliable-mandrill-alternative-smtp-relays/>

Took a quick tour. Mailgun has changed since this lecture was composed.

## Email verification

Verify your id in Mailgun

## Configuration

Modify {application root}/.env file:

In the email server section make the following changes:

# MAIL\_DRIVER=smtp

MAIL\_DRIVER=mailgun

MAILGUN\_DOMAIN=sandbox3f3dca3b7ee94ea8bbf3feeef30b20fc.mailgun.org

MAILGUN\_SECRET=020b315cd84c55d0b87dfeb671f0eb3e-2b0eef4c-a8b44880

# MAIL\_HOST=mailtrap.io

# MAIL\_HOST=smtp.mailgun.org

# MAIL\_PORT=587

# MAIL\_USERNAME=null

# MAIL\_PASSWORD=null

# MAIL\_ENCRYPTION=null

Modify {application root}/config/mail.php:

In the Global "From" Address section make the following changes:

// 'from' => ['address' => null, 'name' => null],

'from' => [

'address' => 'barry.rayfield@gmail.com',

'name' => 'Barry at Gmail'

],

Modify {application root}/config/services.php:

No changes made but the MAIL\_DOMAIN and MAIL\_SECRET env variables were sourced from the mailgun section.

'mailgun' => [

'domain' => env('MAILGUN\_DOMAIN'),

'secret' => env('MAILGUN\_SECRET'),

],

## Sending email part 1 - route

Added a route to {application root}/app/Http/routes:

Route::get('/mail', function () {

$data = [

'title' => 'Hi student I hope you like this course',

'content' => 'This Laravel course was created with ‘

. ‘a lot of love and dedication for you.'

];

Mail::send('emails.test', $data, function ($message)

{

$message->to('barry.rayfield@gmail.com', 'Barry at Gmail')->

subject('Hello student, how are you?');

});

});

Mail:send() is expecting a view. In this case the view test.blade.php in the directory {application root}/resources/views/emails.

So far only an empty file has been created for test.blade.php. It will be fleshed out in the next lecture.

## Sending email part 2 - Finished

Additional requirement "guzzlehttp/guzzle": "~5.3|~6.0"

composer require guzzlehttp/guzzle

Modified test.blade.php:

Used the basic HTML doc layout and added some <h1> and <p> content.

Executed route <http://tutorial.tst/mail/public/mail> resulted in the following error:

Whoops, looks like something went wrong.

1/1 RequestException in CurlFactory.php line 186:

cURL error 60: SSL certificate problem: unable to get local issuer certificate (see <http://curl.haxx.se/libcurl/c/libcurl-errors.html>)

Solution:

Go to {application root}/vendor/guzzlehttp/guxxle/src/client method configureDefaults() and property $defaults

$defaults = [

'allow\_redirects' => RedirectMiddleware::$defaultSettings,

'http\_errors' => true,

'decode\_content' => true,

// 'verify' => true,

'verify' => false,

'cookies' => false

];

More info on Authorized Recipients in Mail Gun: <https://help.mailgun.com/hc/en-us/articles/217531258-Authorized-Recipients>

More info on possibly resolving spam issue: <https://stackoverflow.com/questions/50416923/hotmail-rejecting-all-emails-using-mailgun-or-sparkpost>

More info on possibly resolving spam issue: <https://help.mailgun.com/hc/en-us/articles/360011702394-Why-Do-My-Emails-Go-to-Spam->

More info on possibly resolving spam issue: <https://help.mailgun.com/hc/en-us/articles/202741290-Email-deliverability-tips>

More info on possibly resolving spam issue: <https://stackoverflow.com/questions/43264303/how-to-avoid-spam-in-outlook-hotmail-with-mailgun>

More info on possibly resolving spam issue: <https://stackoverflow.com/questions/48602563/sent-e-mail-from-laravel-goes-to-spam-mailgun>

## Custom domain

Need to upgrade your mailgun account to create a custom domsin.

LDAP login information: try googling ***laravel ldap authentication tutorial***.

LDAP login information link: <https://github.com/Adldap2/Adldap2-Laravel>

For more information on sending mail: <https://laravel.com/docs/5.2/mail>

# Section 26: Git and Github – Version Control

## Git intro

You’re about to learn about GIT and source control

## Git Installation

Repeat of lecture 8. GIT.

## Git commands

Created the directory C:\wwwroot\tutorial\learning-git.

Initialized it for GIT with the command git init.

Typing git by itself will bring up a list of commands you can use with git.

## Git crash lecture

git status get the current status of git under the current project

git add . Add all changes to a list of those to be applied to the repository

git log List all changes applied to a repository

git commit -m "Added feature 1" Commit changes created with add to a repository. –m stands for message. "Added feature 1" is the message used to describe the change in the log file.

git reset –hard {reference} Rollback changes to a specific point in the log. {reference} is a commit object reference which can look similar to: 97de8ead1a488a957e0a039ef4e640ff313c5387

It will appear in the git log

At the moment we are using a local repository.

## Git workflow

git init Initialize repository

Create file/files Do your editing

git add . Queue up files for the repository. “.” is the file skeleton. “.” is all files in current directory.

git status Get the status of files queued up.

git commit -m "added file again 3" Commit queued up changes to repository

git log Get a list of changes that have already been committed.

git reset –hard {reference} Rollback to a specific change level.

git reset {reference} Reset without the “-hard” is a soft reset. This means the follow on changes have now been placed I the queue or staged for a commit.

## Branches

git init Initialize a new repository.

git status Show that you’re on the master branch.

Create files

git add {files} Queue up the edited file for commission to the master branch.

git commit – m ”{description}” Commit the queued up files.

git checkout –b newcategory Create a new branch called “newcategory”

git status Check the status of the new branch and see that you are on the branch “new category”. Messages below:

# On branch newcategory

Git log Check the log and you will see an initial commit where you created the branch.

Create new files or edit existing

Git add {files} Queue up the edited file for commission to the newcategory branch.

git commit – m ”{new description}” Commit the queued up files to the new category branch.

git checkout master Switch back to the master branch.

git status You will see you are on the master branch as per the messages below:

# On branch master

Git checkout newcategory

Switched to branch ‘newcategory’

Git log

Shows two commits. The initial commit to create the branch and the edited files specific to this branch.

Git checkout master Switch back to the master branch.

Git log Only shows the initial commit.

Git merge newcategory Merges the new category branch into the master branch. As per messages below:

Updating {current oject reference}

Fast-forward

{filename} | 0

1 file changed, 0 insertions(+), 0 deletions(-)

Create mode ……. {filename}

Git status Shows on master branch with nothing queued up for commit.

Git log Shows two entries. Initial creation and the merge of the newcategory branch.

Git branch –d newcategory Delete of the no longer needed branch newcategory.

## Creating repositories on Github

Welcome to GITHUB.

Quick tour of GITHUB.

## Git settings

Setting your name and email in GIT.

git config --global user.name "{your name}" Set your name in GIT

git config --global user.email "{valid email}" Set your email address in GIT

Comparision of GitHub vs. GitLab: <https://usersnap.com/blog/gitlab-github/>

Alternative BitBucket: <https://bitbucket.org/product/>

## Cloning and pushing to Github

git clone {url provided by the repository} The repository name will be the new subdirectory name.

git status Check status. See your on master and in sync

On branch master with github

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

git log Check log and see change log that originated with log ion github

commit {object reference} (HEAD -> master, origin/master, origin/HEAD)

Author: Barry Rayfield <barry.rayfield@gmail.com>

Date: Fri Jan 13 01:37:42 2017 -0500

Removed <em> tag from html and place its functionality in the stylesheet

commit {object reference}

Author: Barry Rayfield <barry.rayfield@gmail.com>

Date: Fri Jan 13 01:29:31 2017 -0500

Initial commit of file for the FEND Animal Trading Cards Master

commit {object reference}

Author: brayfiel <barry.rayfield@gmail.com>

Date: Fri Jan 13 01:20:34 2017 -0500

Initial commit

Edit files.

git add . Queue up any changed files for committing.

git status Check status. See local master and remote (GitHub) masters are in sync but there are changes queued up.

On branch master

Your branch is up to date with 'origin/master'.

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: README.md

git commit -m "modified readme.md" Commit changes to local master.

[master a1ea333] modified readme.md

1 file changed, 4 insertions(+), 1 deletion(-)

git push origin master Push changes from local master to remote master (GitHub).

Counting objects: 3, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 375 bytes | 125.00 KiB/s, done.

Total 3 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github.com/brayfiel/fend-animal-trading-cards-master.git

2f13406..a1ea333 master -> master

If the github file that we are cloning requires an SSH key, how do we generate it? Refer to official GitHub documentation, it's nicely explained step by step.

Adding new SSH key to your GitHub account: <https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/>

Generating a new SSH key and adding it to the ssh-agent: <https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>

Connecting to GitHub via SSH: <https://help.github.com/articles/connecting-to-github-with-ssh/>

## Pull request and pulling changes

git pull Will pull updates from the remote site (GitHub) and load them into the local repository.

Discussed forking, cloning to local, editing, local committing, remote push to origin, and creating pull request for one you forked from.

## Contributing to the project

Request access to repository DiazEdwin/codehacking via the Q&A section of this lecture.

Workflow:

1. Forking someone else’s project to your remote GitHub repository,
2. Cloning you project (forked project) to local,
3. Editing local,
4. Local committing,
5. Push to origin (your remote repository) on GitHub
6. Create pull request
7. Forked repository can either accept or reject the pull request.

# Section 27: Application

## Laravel Project Reference Files

Downloaded files: Lecture 187 - laravel-project-online-version.zip

Lecture 187 - laravel-project-local.zip

Lecture 187 - laravel-project-files.zip

Lecture 187 - codehacking-version-5.2.zip

Files have been downloaded and exploded to C:\wwwroot\tutorial\Laravel\_Course\_Solutions

To create the vendor folder run composer update

## New laravel installation

Create new directory with the Laravel core under tutorial called codehacking for this appplication through the use of composer.

Composer create-project laravel/laravel ***codehacking*** 5.2.31 --prefer-dist

***codehacking*** - The name of the project directory.

Don’t forget to modify hosts file, apache \*.conf files, etc. as needed. My installation did not need it.

## Database configuration

Created database codehacking

Ran initial migration with: php artisan migrate

Created 3 tables:

migrations

password\_resets

users

Modified .env

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=***codehacking***

DB\_USERNAME=***root***

DB\_PASSWORD=***mischeif02***

## Setting up views

Create initial login screens: php artisan make:auth

Under {application root}\codehacking\resources\views created the following directories:

admin

Create the following files: index.blade.php

admin\categories

Create the following files: edit.blade.php

index.blade.php

admin\posts

Create the following files: create.blade.php

edit.blade.php

index.blade.php

admin\users

Create the following files: create.blade.php

edit.blade.php

index.blade.php

## Users table Migration

Created local GIT repository to save project files in the {application root}\codehacking directory execute the following:

git init Initialize local repository

git status Show status to confirm hitting local repository and the master branch.

git add . Queue up all modified file for commit.

git commit -m "My first commit - admin views created"

Commit all queued up files and file uner the message: My first commit - admin views created

git log Examine log to confirm commit.

Modify User database migration. Add the following after the id field:

$table->integer('role\_id')->index()->unsigned()->nullable();

$table->integer('is\_active')->default(0);

Create Role migration and model

php artisan make:model Role -m

Modify Role database migration. Add the following after the id field:

$table->string('name');

Additional links on MySQL indicies:

<https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html>

<http://www.treselle.com/blog/mysql-indexes-basicstypes-and-features/>

<https://vanseodesign.com/web-design/the-types-of-indexes-you-can-add-to-mysql-tables/>

## Relation setup and data entry

Ran php artisan migrate to implement the Roles migration.

Ran php artisan migrate:refresh to create role\_id and is\_active columns in Users table.

Added roles administrator, author and subscriber.

Added first user.

Set role\_id to 1 and is\_active to 1.

## Testing relation with Tinker

php artisan tinker

>>> $user = App\User::find(1)

=> App\User {#643

id: 1,

role\_id: 1,

is\_active: 1,

name: "Barry Rayfield",

email: "barry.rayfield@gmail.com",

created\_at: "2019-07-10 21:40:34",

updated\_at: "2019-07-10 21:40:34",

}

>>> $user->role()

=> Illuminate\Database\Eloquent\Relations\BelongsTo {#631}

>>> $user->role

=> App\Role {#647

id: 1,

name: "administrator",

created\_at: "2019-07-10 17:25:00",

updated\_at: "2019-07-10 17:25:00",

}

>>> App\User::create(['name'=>'John Doe'], ['role\_id'=>2], ['is\_active'=>1], ['password'=>'mischeif02'])

=> App\User {#660

name: "John Doe",

updated\_at: "2019-07-10 22:02:26",

created\_at: "2019-07-10 22:02:26",

id: 2,

}

>>> App\User::create(['name'=>'Jane Doe', 'role\_id'=>3, 'is\_active'=>1, 'password'=>'mischeif02', 'email'=>'jane.doe@bsrresearch.com'])

=> App\User {#657

name: "Jane Doe",

role\_id: 3,

email: "jane.doe@bsrresearch.com",

updated\_at: "2019-07-10 22:04:23",

created\_at: "2019-07-10 22:04:23",

id: 3,

}

>>> App\User::create(['name'=>'Ralph Doe', 'role\_id'=>3, 'is\_active'=>1, 'password'=>'mischeif02', 'email'=>'ralph.doe@bsrresearch.com'])

=> App\User {#660

name: "Ralph Doe",

role\_id: 3,

email: "ralph.doe@bsrresearch.com",

updated\_at: "2019-07-10 22:06:03",

created\_at: "2019-07-10 22:06:03",

id: 4,

}

>>> App\User::create(['name'=>'Edna Doe', 'email'=>'edna.doe@bsrresearch.com', 'password'=> bcrypt('mischeif02123456'), 'role\_id'=>3, 'is\_active'=>1]);

=> App\User {#663

name: "Edna Doe",

email: "edna.doe@bsrresearch.com",

role\_id: 3,

updated\_at: "2019-07-10 22:13:02",

created\_at: "2019-07-10 22:13:02",

id: 5,

}

>>> App\User::create(['name'=>'Frank Doe', 'email'=>'frank.doe@bsrresearch.com', 'password'=>bcrypt('mischeif02'), 'role\_id'=>1, 'is\_active'=>1])

=> App\User {#660

name: "Frank Doe",

email: "frank.doe@bsrresearch.com",

role\_id: 1,

updated\_at: "2019-07-10 22:15:39",

created\_at: "2019-07-10 22:15:39",

id: 6,

}

>>>

## Admin controller and routes

Modified Routes.php by adding the line;

Route::resource('admin/users', 'AdminUsersController');

Created the controller AdminUsersController.php with the following;

php artisan make:controller --resource AdminUsersController

## Testing methods

Tested the routes:

+---------+------------------------+-------------------+---------------------------+------------+

| Method | URI | Name | Action | Middleware |

+---------+------------------------+-------------------+---------------------------+------------+

|GET|HEAD |admin/users |admin.users.index |App\Http\Controllers\ | web |

| | | |AdminUsersController@index | |

|GET|HEAD |admin/users/create |admin.users.create |App\Http\Controllers\ | web |

| | | |AdminUsersController@create| |

|GET|HEAD |admin/users/{users} |admin.users.show |App\Http\Controllers\ | web |

| | | |AdminUsersController@show | |

|GET|HEAD |admin/users/{users}/edit|admin.users.edit |App\Http\Controllers\ | web |

| | | |AdminUsersController@edit | |

+---------+------------------------+-------------------+---------------------------+------------+

Modified the views in {Application Root}\app\resources\views\admin\users by adding basic display statements. The views are;

show.blade.php

index.blade.php

create.blade.php

edit.blade.php

With the links;

<http://tutorial.tst/codehacking/public/admin/users>

<http://tutorial.tst/codehacking/public/admin/users/create>

<http://tutorial.tst/codehacking/public/admin/users/1>

<http://tutorial.tst/codehacking/public/admin/users/1/edit>

## Installing NodeJs and files download

Downloaded assets.zip as Lecture\_196\_assets.zip.

Download nodejs from <http://nodejs.org>

Check nodejs version with the command: node -v

As of 7/14/2019 my installed version is: v.10.15.3

Ran npm ls to see a list of the globally installed nodejs applications. Informed there were none installed and asked to do a minor upgrade. Did the upgrade with the commnd:

npm install -g npm

Install gulp with the command: npm install --global gulp

Warning!!! Only use gulp with Laravel 5.2. If you are using Laravel 5.4 you will not be using gulp. You will be using WebPack in its place.

To check Laravel version use the command: php artisan –version

As I am not using XAMPP I had to do the following (based on information from <https://github.com/nodejs/node-gyp#on-windows> );

1. run npm install -g node-gyp

2. Install Microsoft's windows-build-tools (<https://github.com/felixrieseberg/windows-build-tools>) using npm install --global --production windows-build-tools from an elevated PowerShell or CMD.exe (run as Administrator).

3. Here I deviated from the web page by adding the Python 2.7 path to my system path. In this case c:\Users\Barry\.windows-build-tools\python27

4. Reran gulp, received the message about laravel-elixir and reran npm install laravel-elixir --save-dev

5. Run (may have to change order or rerun)

npm install laravel-elixir --save-dev

npm rebuild node-sass

6. Ran gulp. Looks like it worked.

Haven’t used assets.zip yet.

## Gulp and assets

Downloaded assets.zip as Lecture\_197\_assets.zip.

assets.zip from lectures 196 and 197 are the same.

Run npm install to refresh your NPM installation.

From the downloaded file extract the following directories and place them in the following folders:

**Extracted Folder Destination**

css {application root}\resources\assets

js {application root}\resources\assets

fonts {application root}\public

Modify {application root}\gulpfile.js:

elixir(function(mix) {

mix.sass('app.scss')

.styles([

'libs/blog-post.css',

'libs/bootstrap.css',

'libs/font-awesome.css',

'libs/metisMenu.css',

'libs/sb-admin-2.css'

], './public/css/libs.css')

.scripts([

'libs/bootstrap.js',

'libs/jquery.js',

'libs/metisMenu.js',

'libs/sb-admin-2.js',

'libs/scripts.js'

], './public/js/libs.js')

});

.styles([],’’) – for the .css files to compiled and minified

[] – array of .css files to compiled and minified.

‘’ – destination directory and filename of the compiled minified .css file.

.styles([],’’) – for the .js files to compiled and minified

[] – array of .js files to compiled and minified.

‘’ – destination directory and filename of the compiled minified .js file.

The destination directory for both in {application root}.

Example run:

C:\wwwroot\tutorial\codehacking>gulp

[20:42:50] Using gulpfile C:\wwwroot\tutorial\codehacking\gulpfile.js

[20:42:50] Starting 'default'...

[20:42:50] Starting 'sass'...

Fetching Sass Source Files...

- resources\assets\sass\app.scss

**Saving To...**

**- public\css\app.css**

[20:42:51] Finished 'default' after 1.65 s

[20:42:51] gulp-notify: [Laravel Elixir] Sass Compiled!

[20:42:51] Finished 'sass' after 1.74 s

[20:42:51] Starting 'styles'...

Fetching Styles Source Files...

- resources\assets\css\libs\blog-post.css

- resources\assets\css\libs\bootstrap.css

- resources\assets\css\libs\font-awesome.css

- resources\assets\css\libs\metisMenu.css

- resources\assets\css\libs\sb-admin-2.css

**Saving To...**

**- ./public/css/libs.css**

[20:42:51] gulp-notify: [Laravel Elixir] Stylesheets Merged!

[20:42:51] Finished 'styles' after 83 ms

[20:42:51] Starting 'scripts'...

Fetching Scripts Source Files...

- resources\assets\js\libs\bootstrap.js

- resources\assets\js\libs\jquery.js

- resources\assets\js\libs\metisMenu.js

- resources\assets\js\libs\sb-admin-2.js

- resources\assets\js\libs\scripts.js

**Saving To...**

**- ./public/js/libs.js**

[20:42:52] gulp-notify: [Laravel Elixir] Scripts Merged!

[20:42:52] Finished 'scripts' after 858 ms

C:\wwwroot\tutorial\codehacking>

## Admin master file - download file

Downloaded admin.blade.php.zip as Lecture\_198\_admin.blade.php.zip.

Extracted it to {application root}\resources\views\layouts

Had to modify some of the href=’’ references to get the layout to work. Due to my setup being different from class.

Originally: <a class="navbar-brand" href="/">Home</a>

Edited: <a class="navbar-brand" href="./">Home</a>

Originally: <a href="/users">All Users</a>

Edited: <a href="./admin/users">All Users</a>

Originally: <a href="/users/create">Create User</a>

Edited: <a href="./admin/users/create">Create User</a>

Two sections have been defined for custom content. They are:

@yield('content')

@yield('footer')

Use the asset() function in admin.blade.php to utilize the .css and .js files generated by gulp:

<link href="{{asset('css/app.css')}}" rel="stylesheet">

<link href="{{asset('css/libs.css')}}" rel="stylesheet">

<script src="{{asset('js/libs.js')}}"></script>

Create {application root}\resources\views\admin\index.blade.php. It should work in tandem with {application root}\resources\views\layouts\admin.blade.php.

@extends('layouts.admin');

@section('content')

<h1>Admin Index</h1>

@endsection

Modify routes.php to test {application root}\resources\views\layouts\admin.blade.php and {application root}\resources\views\admin\index.blade.php.

Route::get('/admin', function () {

return view('admin.index');

});

Open browser and test with <http://tutorial.tst/codehacking/public/admin>

## Version control and modifying master page

Modified {application root}\resources\assets\sass\app.scss in order to make a change to the styling of the admin page. Eliminate the excess space at the top. Added the following;

#admin-page {

padding-top: 0px;

}

Once modified and saved gulp was rerun to update the .css files.

At this point it’s time to update your local git repository. Run the following;

git add . queue up all new, modified and deleted files to update the repository

git status show a list of the queued up files.

git commit -m "Admin master page setup" Commit the changes to the repository.

git log Show a list of the committed changes.

## Displaying users

Modified controller {application root}\app\Http\Controllers\AdminUserController.php to extract user data and pass it to the view.

<?php

namespace App\Http\Controllers;

***use App\User;*** // added to allow access to the

// User::xxx methods

use Illuminate\Http\Request;

use App\Http\Requests;

class AdminUsersController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

***$users = User::All();*** // call to the database to request all

// user table data

return view('admin.users.index'***, compact('users')***);

// modified call to the view to pass

// $users variable to the view

}

Modify modify view {application root}\resources\views\admin\users\index.blade.php to display the data in a table.

@extends('layouts.admin');

@section('content')

<h1>Users</h1>

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>Email</th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

***@if($users)***

***@foreach($users as $user)***

***<tr>***

***<td>{{$user->id}}</td>***

***<td>{{$user->name}}</td>***

***<td>{{$user->email}}</td>***

***<td>{{$user->created\_at->diffForHumans()}}</td>***

***<td>{{$user->updated\_at->diffForHumans()}}</td>***

***</tr>***

***@endforeach***

***@endif***

</tbody>

</table>

@endsection

HTML template was copied from a basic table example from w3schools for bootstrap in css, <https://www.w3schools.com/bootstrap/bootstrap_tables.asp>

Bold and italic section actually uses the passed data

## Create page

Modified All Users view {application root}\resources\views\admin\users\index.blade.php and added columns for role and active.

In <thead> added:

<th>Role</th>

<th>Active</th>

In <tbody> added:

<td>{{$user->role->name}}</td>

<td>{{$user->is\_active == 1 ? 'Active' : 'Not Active'}}</td>

Started to create the view {application root}\resources\views\admin\users\create.blade.php

## Laravel collective html package

See Lectures “132. UPDATE – IMPORTANT” and “133. Installing Package And Testing” for installation instructions.

See <https://packagist.org/packages/laravelcollective/html#v5.2.6> for installation requirements.

Modified the view {application root}\resources\views\admin\users\create.blade.php

Added a basic form using the Laravel Collective HTML syntax.

{!! Form::open(['method'=>'POST', 'action'=>'AdminUsersController@store']) !!}

<div class="form-group">

{!! Form::label('title', 'Title:') !!}

{!! Form::text('title', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::submit('Create User', ['class'=>'btn btn-primary']) !!}

</div>

{!! Form::close() !!}

## Testing form and creating form fields

Modified the view {application root}\resources\views\admin\users\create.blade.php

Renamed the label and text fields to ‘name’ and ‘Name:’ from ‘title’ and ‘Title:’ respectively.

Renamed the submit button to ‘Create User’.

Added three more label and text fields to support Email, Role and Status.

Modified the store method of the controller AdminUserController to display the contents of $request parameter for testing purposes.

return $request->all();

## Working on user form status field

Modified the view {application root}\resources\views\admin\users\create.blade.php

Changed status from a text field to a drop down with the following:

{!!

Form::***select***(

'status',

***array(1=>'Active', 0=>'Not Active'),***

***0,***

['class'=>'form-control']

)

!!}

select – tells laravel collective html to generate a dropdown box.

array() – tells the select statement the options in the dropdown box.

0 – tells the select statement what the delfault value for the dropdown box is.

## Populating the user roles select

Changed Role to a dropdown based on the data in the roles table.

Modified create() method in controller AdminUserController

$roles = Role::lists('name', 'id')->all();

return view('admin.users.create', compact('roles'));

Grabbed name and id from the roles table for all records and pass it to the view via the compact() function..

The list() method creates an associative array to hold he data. The values for the array is parameter 1 (‘name’) and the key for the array is parameter 2 (‘id’).

***WARNING!!! It should be noted that list() has been deprecated. It’s replacement is pluck().***

Modified the view {application root}\resources\views\admin\users\create.blade.php

{!!

Form::select(

'Role\_id',

***[''=>'Choose Options'] + $roles,***

null,

['class'=>'form-control']

)

!!}

[''=>'Choose Options'] + $roles – Concatenation of arrays. First array with an index of ‘ ’ has a text prompt of ‘Choose Options’. The second array contains the roles data from the controller AdminUserController. The + concatenates them.

## Password field and custom request

Modified the view {application root}\resources\views\admin\users\create.blade.php to add a password field:

<div class="form-group">

{!! Form::label('password', 'Password:') !!}

{!! Form::password('password', ['class'=>'form-control']) !!}

</div>

Created a validation request ({application root}\app\Http\Requests\UsersRequest.php) which feeds the controller AdminUsersController with the command:

php artisan make:request UsersRequest

In UsersRequest set the authorize() method to return true.

In UsersRequest set the rules() method to:

public function rules()

{

return [

'name'=>'required',

'email'=>'required',

'role\_id'=>'required',

'is\_active'=>'required',

'password'=>'required'

];

}

Modified store() method in controller AdminUserController:

At the top added:

use App\Http\Requests\UsersRequest;

Modified the store() method;

public function store(***UsersRequest $request***)

{

return $request->all();

}

UserRequest was originally Request but was changed to handle the results from the validation rules.

## Displaying errors and including with blade

Error are stored in a global called $error.

Created a routine to display those errors. As they will be used in a number of places the routine will be turned into an include called form\_error.blade.php and stored in the folder {application root}\resources\views\includes:

@if(count($errors) >0)

<div class="alert alert-danger">

<ul>

@foreach($errors->all() as $error)

<li>{{$error}}</li>

@endforeach

</ul>

</div>

@endif

The view create.blade.php was modified by adding the following statement at the end just above @endsection:

@include('includes.form\_error');

## Adding upload file feature to form

Modified create.blade.php

Altered Form statement to handle files (“multipart forms” or enctype="multipart/form-data">):

{!!

Form::open(

[

'method'=>'POST',

'action'=>'AdminUsersController@store',

***'files'=>true***

]

)

!!}

After is\_active dropdown add the code below to select a file:

<div class="form-group">

{!! Form::label('file', 'File:') !!}

{!! Form::file('file', null, ['class'=>'form-control']) !!}

</div>

Created migration 2019\_07\_18\_050725\_add\_photo\_id\_to\_users.php to add a photo\_id field to the users table.

Modified model Users.php to add photo\_id to $fillable property.

## Persisting data and fixing bug

Modify store() method in {application root}\app\Http\Controllers\AdminUserController.php

public function store(UsersRequest $request)

{

// return $request->all();

User::create($request->all());

return redirect('/admin/users');

}

Even though, the $request is trying a file value for which there is no field Laravel will not flag it with an error. It will just process the other fields.

Already fixed the bugs he was referring to.

## User photos migration - relation - mass-assignment

Create the migration and model for the Photos table

php artisan make:model Photo –m

Should have been done as a first step but now is a good time to commit changes to the git repository.

Queue up changes to commit:

git add .

Commit those changes to the database.

git commit -m "users able to persist”

Modify migration {application root}\database\migrations\2019\_07\_18\_154702\_create\_ photos\_table.php

Schema::create('photos', function (Blueprint $table) {

$table->increments('id');

***$table->string('file');***

$table->timestamps();

});

Added the column file to hold the path to and name of the photo.

Modify model {application root}\app\Photo.php

Added the property $fillable

protected $fillable = ['file'];

Modify model {application root}\app\User.php

Added the many(Users) to one(Photos) relationship.

public function photo()

{

return $this->belongsTo('App\Photo');

}

Applied the migration:

php artisan migrate

This will have to be revisited in the future. When the photos table is used by posts you will possibly have a many to many relationship and the logic will have to be reworked.

## Creating links and more

Modified the view admin.blade.php in the folder {application root} \resources\views\layouts:

<li>

{{-- <a href="/codehacking/public/admin/users">All Users</a> --}}

<a href="{{route('admin.users.index')}}">All Users</a>

</li>

<li>

{{-- <a href="/codehacking/public/admin/users/create">Create User</a> --}}

<a href="{{route('admin.users.create')}}">Create User</a>

</li>

The lines encased in {{-- --}} are the original lines.

routes() – Let’s you use the routes you created in routes.php instead of having to code an actual url (full or relative). This way, if a route changes you make the change in route.php and you’re done. As opposed to searching for all the places you might have used the url.

Modified AdminUsersController.php. This was just to prove we are received the photo\_id.

public function store(UsersRequest $request)

{

// return $request->all();

// User::create($request->all());

// $input = $request->all();

if ($request->file('photo\_id')) {

return "photo exists";

}

}

Difference between {{ }} and {!! !!}

{‌{$string}} Escapes DATA - this will show the contents of $string escaped

{!! $string!!} DOES NOT escape data - this will show the text $string

Example:

$name = "<b>Jeff</b>";

{‌{ $name }} displays “<b>Jeff</b>”

{!! $name !!} displays in bold “**Jeff”**

This is a blade function.

## Persisting user with photo part 1

Modified the store() method of the controller AdminUsersController.

public function store(UsersRequest $request)

{

$input = $request->all();

if ($file = $request->file('photo\_id')) {

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

$photo = Photo::create(['file'=>$name]);

$input['photo\_id'] = $photo->id;

}

}

## Persisting user with photo part 2

Added the following at the top:

use App\Photo;

Modified store() method of controller AdminUsersController

public function store(UsersRequest $request)

{

$input = $request->all();

if ($file = $request->file('photo\_id')) {

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

$photo = Photo::create(['file'=>$name]);

$input['photo\_id'] = $photo->id;

}

$input['password'] = bcrypt($request->password);

User::create($input);

return redirect('/admin/users');

}

It’s working!!! Time to commit it.

Queue up all the changes:

git add .

Commit the changes:

git commit –m “Users with uploading feature working”

## Displaying photos using an accessor

Modified the model Photo.php to create property to hold the location of the images directory.

protected $uploads = '/codehacking/public/images/';

And created the accessor to provide the full path and file name of the photo.

public function getFileAttribute($photo)

{

return $this->uploads . $photo;

}

Modified the admin users index view index.blade.php to add a photo column heading and add the <img> tag for the photo to the data section of the table.

. . .

<th>Photo</th>

. . .

. . .

<td>

<img

height="50"

src="{{$user->Photo ? $user->Photo->file : ''}}"

alt="Not Avail."

>

</td>

. . .

## Edit users part 1 - setting up the form

Modify the controller AdminUsersController.php

public function edit($id)

{

$user = User::findOrFail($id);

$roles = Role::lists('name', 'id')->all();

return view('admin.users.edit', compact('user', 'roles'));

}

Copied the view {application root}\resources\views\admin\users\create.blade.php to {application root}\resources\views\admin\users\edit.blade.php

Modified the view {application root}\resources\views\admin\users\edit.blade.php

Changed Form::open to Form::model to use model binding under Laravel Collective HTML

{!!

Form::model(

$user,

[

'method'=>'PATCH',

'action'=>['AdminUsersController@update', $user->id],

'files'=>true

]

)

!!}

Changed the <h1>:

<h1>Edit Users</h1>

Removed the initial option from the roles dropdown.

<div class="form-group">

{!! Form::label('role\_id', 'Role:') !!}

{!!

Form::select('role\_id', $roles, null, ['class'=>'form-control'])

!!}

</div>

Form works but not 100%. See part 2.

## Edit user part 2 - displaying images and status

Modified the view {application root}\resources\views\admin\users\edit.blade.php

Fixed Active/Not Active not being reflected in the view:

{!!

Form::select(

'is\_active',

array(1=>'Active', 0=>'Not Active'),

***null,***

['class'=>'form-control'])

!!}

By not providing a default value via the use of null it now picks up the value of the passed is\_active parameter (field).

Added display of photo:

Wrapped form in col-sm-9

Preceded that with a div with a class of col-sm-3 to display the photo.

In that div placed an <img> tag to display the photo

<img

src=

"{{$user->photo ?

$user->photo->file :

'https://via.placeholder.com/400'

}}"

alt="Not On File"

class="img-responsive img-rounded"

>

Modified the view {application root}\resources\views\admin\users\index.blade.php to show a dummy image if a real one is not available.

<td>

<img

height="50"

src=

"{{$user->Photo ?

$user->Photo->file :

***'https://via.placeholder.com/50'}}"***

alt="Not Avail."

>

</td>

## Updating part 1 and displaying errors

Made some cosmetic changes to the view {application root}\resources\views\admin\users\ edit.blade.php to show errors properly

Modified the update() method of the controller AdminUsersController.php to prep mfor storing the updated record:

public function update(UsersRequest $request, $id)

{

$user = User::findOrFail($id);

$input = $request->all();

if ($file = $request->file('photo\_id')) {

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

}

return $request->all();

}

## Updating part 2

As not all fields are required to be entered or updated in edit, ie: password, a new request was created called {application root}\app\Http\Requests\UsersEditRequest.php.

php artisan make:request UsersEditRequest

In the request {application root}\app\Http\Requests\UsersEditRequest.php the authorize() method was set to return true;

The rules() method was set to return the following;

return [

'name'=>'required',

'email'=>'required',

'role\_id'=>'required',

'is\_active'=>'required',

];

Modified the update() method of the controller AdminUsersController.php to the following:

public function update(***UsersEditRequest*** $request, $id)

{

$user = User::findOrFail($id);

$input = $request->all();

if ($file = $request->file('photo\_id')) {

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

***$photo = Photo::create(['file'=>$name]);***

***$input['photo\_id'] = $photo->id;***

}

***$user->update($input);***

***return redirect('/admin/users');***

}

Tested it, it seems to work.

## Updating part 3 - Fixing loose ends

Modify the store() method of AdminUsersController.php

Replaced statement at the top:

$input = $request->all();

With:

if (trim($request->password) == '') {

$input = $request->except('password');

} else {

$input = $request->all();

$input['password'] = bcrypt($request->password);

}

And removed at the bottom:

$input['password'] = bcrypt($request->password);

Modify the update() method of AdminUsersController.php

Replaced:

$input = $request->all();

With the following:

if (trim($request->password) == '') {

$input = $request->except('password');

} else {

$input = $request->all();

$input['password'] = bcrypt($request->password);

}

## Security part 1 – middleware registration

Only those who are logged in, administrators and active should be able to get to <http://tutorial.tst/codehacking/public/admin> and the links below it.

To accomplish this start by creating the middleware Admin.php in the folder {application root}\app\Http\Middleware

php artisan make:middleware Admin

Did not update it yet,

Committed all updates (git) to the repository before continuing.

Modified the property $routeMiddleware in the {application group}\app\Http\Kernel.php to register the new middleware. Added the statement below to the end of the property:

'admin' => \App\Http\Middleware\Admin::class,

Modified {application group}\app\Http\Routes.php to wrap the Route::resource with a Route::group as shown below.

Route::group(['middleware'=>'admin'], function () {

Route::resource('admin/users', 'AdminUsersController');

});

For additional information on middleware see: <https://laravel.com/docs/5.2/middleware>

## Security part 2 - middleware - custom method and 404 page

Modified middleware class Admin.php

Added reference to Auth at the top of the file:

use Illuminate\Support\Facades\Auth;

Modified the handle() method to look as follows:

public function handle($request, Closure $next)

{

if (Auth::check()) {

if (Auth::user()->isAdmin()) {

return $next($request);

}

}

return redirect('/');

}

Created a custom 404 error page:

Went to {application root}\resources\views\errors

Created 404.blade.php. 404 is the HTTP error code

At this point all the file contains is:

<h1>Custom 404 Page</h1>

Modified the model User.php and added the isAdmin() method:

public function isAdmin()

{

if ($this->role->name == "administrator") {

return true;

}

return false;

}

Testing is put off to the next lecture.

## Deleting users

Tested middleware from previous lecture, it worked.

Modified the view {application root}\resources\views\layouts\admin.blade.php to add the logged in user’s name to the top of the screen. Made the modification in bold:

<a class="dropdown-toggle" data-toggle="dropdown" href="#">

<i class="fa fa-user fa-fw"></i>

***{{ Auth::user()->name }}***

<i class="fa fa-caret-down"></i>

</a>

Modified the view {application root}\resources\views\admin\users\edit.blade.php to add a delete button to the screen. See changes in bold:

<div class="form-group">

{!! Form::submit('Update User', ['class'=>'btn btn-primary ***col-sm-6***']) !!}

</div>

{!! Form::close() !!}

***{!! Form::open(['method'=>'DELETE',***

***'action'=>['AdminUsersController@destroy', $user->id]])***

***!!}***

***<div class="form-group">***

***{!! Form::submit('Delete User', ['class'=>'btn btn-danger col-sm-6']) !!}***

***</div>***

***{!! Form::close() !!}***

Modified the destroy() method in the controller AdminUsersController.php

public function destroy($id)

{

User::findOrFail($id)->delete();

return redirect('/admin/users');

}

## Some flash messages

Goal is to store a message in the session variable upon a successful delete and display it on the index view. To accomplish this:

Modify the controller AdminUsersController.php.

At the top add:

use Illuminate\Support\Facades\Session;

Modify the destroy() method. Modifications in bold:

public function destroy($id)

{

User::findOrFail($id)->delete();

***Session::flash('deleted\_user', 'The user has been deleted');***

return redirect('/admin/users');

}

session is a Laravel class for manipulation the session array/settings.

flash(key, value) is the method to store a value under a key in the session array.

Modify the view {application root}\resources\views\admin\users\index.blade.php.

Above <h1>Users</h1> insert the following:

@if(Session::has('deleted\_user'))

<p class="bg-danger text-center">{{session('deleted\_user')}}</p>

@endif

If the key 'deleted\_user' exists in the session array display the message stored under it.

Alternate method using redirect()->with():

Modify the controller AdminUsersController.php.

$user = User::findOrFail($id);

if ($user->delete()) {

***return redirect('/admin/users')->with('success',***

***'User '.$user->name.' deleted ');***

} else {

***return redirect('/admin/users')->with('failure',***

***'User '.$user->name.' not deleted');***

}

with() performs the same function as Session::flash()

For more information see: <https://laravel.com/docs/5.2/responses#redirecting-with-flashed-session-data>

Modify the view {application root}\resources\views\admin\users\index.blade.php as detailed above.

## Deleting images from the directory

Goal: To delete any image (single) that is associated with this user.

Modify the destroy method() in controller AdminUsersController.php.

public function destroy($id)

{

$user = User::findOrFail($id);

***if($user->photo) {***

***$photo = strstr($user->photo->file, '/images/');***

***unlink(public\_path() . $photo);***

***}***

$user->delete();

Session::flash('deleted\_user', 'The user has been deleted');

return redirect('/admin/users');

}

Test to see if a photo is associated with the user.

If there is add the public\_path() to the photo name and delete (unlink()) it.

Normally you would have

***unlink(public\_path() . $user->photo->file);***

but my installation is out of normal so you have the string manipulation

***$photo = strstr($user->photo->file, '/images/');***

## Adding more security

Goal: Added an is\_active conditional to the isAdmin() decision block.

In the user model modify the isAdmin() method as shown below in bold:

public function isAdmin()

{

if ($this->role->name == "administrator" ***&& $this->is\_active == 1***) {

return true;

}

return false;

}

Goal: Once successfully logged in redirect all to the admin page.

Modify the property $redirectTo in the controller {application root}\app\Http\Controllers\ Auth\AuthController.php. Change it’s value to:

protected $redirectTo = '***/admin***';

Now all users who are successfully logged in will go to the admin page.

# Section 28: Application - Posts

## Setting route files

Modified 404 page not found error page ({application root}\resources\views\errors\ 404.blade.php)

@extends('layouts.app')

@section('content')

<h1 class="text-center">Opps No Page Available</h1>

@endsection

Modified routes.php adding the posts controller (AdminPostsController). Changes in bold.

Route::group(['middleware'=>'admin'], function () {

Route::resource('admin/users', 'AdminUsersController');

***Route::resource('admin/posts', 'AdminPostsController');***

});

Created {application root}\app\Http\Controllers\AdminPostsControllers.

php artisan make:controller --resource AdminPostsController

Modified index() and create() methods of {application root}\app\Http\Controllers\ AdminPostsControllers.

public function index()

{

***return view('admin.posts.index');***

}

public function create()

{

***return view('admin.posts.create');***

}

Modified {application root}\resources\views\layouts\admin.blade.php to add the new routes for posts.

<li>

***{{-- <a href="./posts">All Posts</a> --}}***

<a href="***{{route('admin.posts.index')}}***">All Posts</a>

</li>

<li>

<a href="***{{route('admin.posts.create')}}***">Create Post</a>

</li>

Modified {application root}\resources\views\admin\posts\index.blade.php to test basic connectivity.

@extends('layouts.admin');

@section('content')

<h1>Posts Index - Edited</h1>

@endsection

Modified {application root}\resources\views\admin\posts\create.blade.php to test basic connectivity.

@extends('layouts.admin');

@section('content')

<h1>Posts Create - Edited</h1>

@endsection

Successfully tested connectivity to index and create.

## Migration

Create model and migration for Posts table.

php artisan make:model Post –m

Created model Post.php

Created migration 2019\_08\_01\_220052\_create\_posts\_table.php

Modified 2019\_08\_01\_220052\_create\_posts\_table.php by adding the fields in bold:

Schema::create('posts', function (Blueprint $table) {

$table->increments('id');

$table->integer('user\_id')->unsigned()->index();

$table->integer('category\_id')->unsigned()->index();

$table->integer('photo\_id')->unsigned()->index();

$table->string('title');

$table->text('body');

$table->timestamps();

});

Ran the migration and created the Posts table.

php artisan migrate

Successfully created Posts table.

## Displaying post

Modified model Post.php

protected $fillable = [

'category\_id',

'photo\_id',

'title',

'body'

];

Don’t forget to save it and then enter tinker or if you’re already in tinker exit and re-enter it.

Enter Tinker.

php artisan tinker

Use Tinker to create a post in the database.

$post = App\Post::create([

'title'=>'My First Post',

'body'=>'I love Laravel with Edwin Diaz'

])

New post created.

Modified index() method of AdminPostsController

public function index()

{

$posts = Post::All();

return view('admin.posts.index', compact('posts'));

}

Modified {application root}\resources\views\admin\posts\index.blade.php to list the posts in the database.

@extends('layouts.admin');

@section('content')

@if(Session::has('deleted\_user'))

<p class="bg-danger text-center">{{session('deleted\_user')}}</p>

@endif

<h1>Posts</h1>

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>User</th>

<th>Category</th>

<th>Photo</th>

<th>Title</th>

<th>Body</th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

@if($posts)

@foreach($posts as $post)

<tr>

<td>{{$post->id}}</td>

<td>{{$post->user\_id}}</td>

<td>{{$post->category\_id}}</td>

<td>{{$post->photo\_id}}</td>

<td>{{$post->title}}</td>

<td>{{$post->body}}</td>

<td>{{$post->created\_at->diffForHumans()}}</td>

<td>{{$post->updated\_at->diffForHumans()}}</td>

</tr>

@endforeach

@endif

</tbody>

</table>

@endsection

Tested it, it works. All Posts (index) now brings up a listing of posts in the database.

## Relationship setup

Post to User tables

Modify User Model (1 to many relationship):

public function posts()

{

return $this->hasMany('App\Post');

}

Modify Post Model (many to 1 relationship):

public function user()

{

return $this->belongsTo('App\User');

}

Modify admin.posts.index(In bold):

<th>Id</th>

***<th>Owner</th>***

<th>Category</th>

. . .

. . .

<td>{{$post->id}}</td>

***<td>{{$post->user->name}}</td>***

<td>{{$post->category\_id}}</td>

Extend the same logic to the photo and category (to be created later) tables:

Modify Post Model (many to 1 relationships):

public function photo()

{

return $this->belongsTo('App\Photo');

}

public function category()

{

return $this->belongsTo('App\Category');

}

## Creating form part 1

Worked on the web page.

Modified the view admin.posts.create

@extends('layouts.admin');

@section('content')

<h1>Create Posts</h1>

{!! Form::open([

'method'=>'POST',

'action'=>'AdminPostsController@store',

'files'=>true

]) !!}

<div class="form-group">

{!! Form::label('title', 'Title:') !!}

{!! Form::text('title', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::label('category\_id', 'Category:') !!}

{!! Form::select(

'category\_id',

array(''=>'options'),

null,

['class'=>'form-control']

) !!}

</div>

<div class="form-group">

{!! Form::label('photo\_id', 'Photo:') !!}

{!! Form::file('photo\_id', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::label('body', 'Description:') !!}

{!! Form::textarea('body', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::submit('Create Post', ['class'=>'btn btn-primary']) !!}

</div>

{!! Form::close() !!}

@endsection

## Creating form part 2

Confirmed what was stored in $request in the store() method of the controller AdminPostsController.

Create the data validation request {application root}\app\Http\Requests\ PostsCreateRequest.php.

php artisan make:request PostsCreateRequest

Modify the request PostsCreateRequest with the following:

public function rules()

{

return [

'category\_id' =>'required',

'photo\_id' =>'required',

'title' =>'required',

'body' =>'required'

];

}

Modified the view admin.posts.create.

Place the existing <form> in a <div> with a class of row.

After the <form> add a <div> with a class of row for the error messages include.

<div class="row">

@include('includes.form\_error');

</div>

## Creating form part 3

Modify the request PostsCreateRequest with the following to add custom error messages:

public function messages()

{

return [

'title.required' => 'Error!!! A title is required.',

'category\_id.required' => 'Error!!! A category must be selected.',

'photo\_id.required' => 'Error!!! A photo is required.',

'body.required' => 'Error!!! The desc. cannot be left blank.',

];

}

Modify the controller AdminPostsController.

use App\Http\Requests\PostsCreateRequest;

. . .

. . .

public function store(PostsCreateRequest $request)

{

//return $request->all();

$user = Auth::user();

$user

}

## Creating post 1

Modified the request PostsCreateRequests.php. Commented out the category\_id rule for now.

Modified the store() method of AdminPostsController.php

use Auth;

. . .

. . .

public function store(PostsCreateRequest $request)

{

$input = $request->all();

$user = Auth::user();

if ($file = $request->file('photo\_id')) {

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

$photo = Photo::create(['file'=>$name]);

$input['photo\_id'] = $photo->id;

}

}

## Creating post 2

Modified the store() method of AdminPostsController.php. See modification in ***bold***.

public function store(PostsCreateRequest $request)

{

$input = $request->all();

$user = Auth::user();

if ($file = $request->file('photo\_id')) {

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

$photo = Photo::create(['file'=>$name]);

$input['photo\_id'] = $photo->id;

}

***$user->posts()->create($input);***

***return redirect('/admin/posts');***

}

Successfully tested the create posts routine.

## Displaying images in post

Modified view admin.posts.create to provide an array for category. See changes in ***bold***.

<div class="form-group">

{!! Form::label('category\_id', 'Category:') !!}

{!!

Form::select(

'category\_id',

***array('1'=>'PHP','2'=>'JavaScript'),***

null,

['class'=>'form-control']

)

!!}

</div>

Modified view admin.posts.index to display photos instead of photo\_id. See changes in ***bold.***

<td>

***<img***

***height="50"***

***src="{{$post->Photo ? $post->Photo->file :***

***'https://via.placeholder.com/50'}}"***

***alt="Not Avail."***

***>***

</td>

## Creating model and migration for categories

At this time save work to GIT

Queue up changes to be saved.

git add .

Move changes into the repositories.

git commit -m 'able to create post with static category'

Create migration and model for the Category table

php artisan make:model Category -m

Entered data into the categories table to create PHP, Laravel, Javascript, and Bootstrap.

## Displaying and creating posts with categories

Modified the view admin.posts.index.

<td>{{$post->category ? $post->category->name : "Uncategorized" }}</td>

Modify create() method in AdminPostsController. Changes in ***bold***.

public function create()

{

***$categories = Category::lists('name', 'id')->all();***

return view('admin.posts.create'***, compact('categories')***);

}

lists has been deprecated. So change lists to pluck. For example:

public function create()

{

$categories = Category::pluck('name','id')->all();

return view('admin.posts.create',compact('categories'));

}

For more information see: <https://laravel.com/docs/5.2/collections#method-pluck>

Modified the view admin.posts.create. See changes in ***bold***.

<div class="form-group">

{!! Form::label('category\_id', 'Category:') !!}

{!!

Form::select(

'category\_id',

***[''=>'Choose Category'] + $categories,***

null,

['class'=>'form-control']) !!}

</div>

## Let's stop and push to GitHub

Create repository on github.

Queue up all changes to this point: git add .

Commit changes to the local repository.

git commit -m ' able to create posts with dynamic categories'

Connect to remote repository.

git remote add origin <https://github.com/brayfiel/codehacking.git>

Push the local repository up to github.

git push -u origin master

Messages:

Counting objects: 325, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (299/299), done.

Writing objects: 100% (325/325), 31.11 MiB | 2.40 MiB/s, done.

Total 325 (delta 126), reused 0 (delta 0)

remote: Resolving deltas: 100% (126/126), done.

To https://github.com/brayfiel/codehacking.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

Check github repository brayfiel/codehacking and see all code (except what is listed in .gitignore) has been uploaded.

## Pushing to GitHub warning

Reminder that files listed in .gitignore will not be uploaded to git.

If you want, you can remove entries from .gitignore if you want a more complete upload to git.

The one entry not to delete is .env. It contains id and password for your database and other environment information custom to your site.

When you recreate .env on a new site you will have recreate (generate) a new “APP\_KEY”. That is done with the command:

php artisan key: generate

When you clone your project to a new location you would follow these steps to get it working;

1. Clone/Download the repository on your disk.

2. Open the project in you choice of IDE or editor.

3. Rename '.env.example' to '.env' and provide your database information in that file(dbname,username,password)

4. Open console and change directory to your project

5. Run composer install

6. Run npm install (if neccessary)

7. Run php artisan key:generate

8. Run php artisan migrate

That should do it.

## Editing part 1 - setting up the page and form

Modify view admin.posts.index to make the author name a lot link to edit a post.

<td>

<a href="{{route('admin.posts.edit', $post->id)}}">

{{$post->user->name}}

</a>

</td>

Modify edit method of the controller AdminPostsController.

public function edit($id)

{

$post = Post::findOrFail($id);

$categories = Category::lists('name', 'id')->all();

return view('admin.posts.edit', compact('post', 'categories'));

}

Load current post

Load all categories

Go to view admin.posts.edit with the post and the category array.

Modify the view admin.posts.edit

@extends('layouts.admin');

@section('content')

<h1>Edit Posts</h1>

<div class="row">

{!!

Form::model(

$post,

[

'method'=>'PATCH',

'action'=>['AdminPostsController@update', $post->id],

'files'=>true

]

)

!!}

<div class="form-group">

{!! Form::label('title', 'Title:') !!}

{!! Form::text('title', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::label('category\_id', 'Category:') !!}

{!!

Form::select(

'category\_id', $categories, null, ['class'=>'form-control']

)

!!}

</div>

<div class="form-group">

{!! Form::label('photo\_id', 'Photo:') !!}

{!! Form::file('photo\_id', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::label('body', 'Description:') !!}

{!! Form::textarea('body', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::submit('Save Post', ['class'=>'btn btn-primary']) !!}

</div>

{!! Form::close() !!}

</div>

<div class="row">

@include('includes.form\_error');

</div>

@endsection

## Editing part 2 - Lets edit the post

Modify edit method of the controller AdminPostsController.

public function update(Request $request, $id)

{

$input = $request->all();

if ($file = $request->file('photo\_id')) {

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

$photo = Photo::create(['file'=>$name]);

$input['photo\_id'] = $photo->id;

}

Auth::user()->posts()->whereId($id)->first()->update($input);

return redirect('/admin/posts');

}

Discussed Laravel helper functions available to developers. For more information see: <https://laravel.com/docs/5.2/helpers>

## Deleting - Challlenge

Modified the view admin.posts.edit to add a delete button. See changes in ***bold***.

<div class="form-group">

{!! Form::submit('Update Post', ['class'=>'btn btn-primary ***col-sm-6***']) !!}

</div>

{!! Form::close() !!}

***{!!***

***Form::open([***

***'method'=>'DELETE',***

***'action'=>['AdminPostsController@destroy', $post->id]***

***])***

***!!}***

***<div class="form-group">***

***{!! Form::submit('Delete Post', ['class'=>'btn btn-danger col-sm-6']) !!}***

***</div>***

***{!! Form::close() !!}***

Challenged to write the destroy method for the adminpostscontroller.

Cloned and then edited the routine from the destroy method of the AdminUsersController.

Tested, it works. Now we’re off to see the official solution.

## Deleting - Solution

Official solution similar to my solution. My solution added more error checking.

Modified the method destroy in the controller AdminPostsController.

public function destroy($id)

{

$post = Post::findOrFail($id);

if ($post->photo\_id) {

$photoFileName = strstr($post->photo->file, '/images/');

unlink(public\_path() . $photoFileName);

$photo = Photo::whereId($post->photo\_id);

$photo->delete();

}

$post->delete();

Session::flash('deleted\_user', 'The user has been deleted');

return redirect('/admin/posts');

}

Retrieve the post.

Delete the photo attached to the post if it exists.

Delete the post.

Go to the index page.

## Deleting users with relations

Modified Posts DATABASE MIGRATION.

Added foreign key constraint to cascade deletes on the deletion of a user\_id.

$table->foreign('user\_id')->references('id')->on('users')->onDelete('cascade');

Issued the command php artisan migrate:refresh to wiep existing data, recreate the databases and apply the new constraint.

Deleted the photos in the images directory as well.

Register new users.

Recreate roles and categories with phpMyAdmin.

Edit Users to add phptos and make them active.

Create a few posts for a user.

Delete the user.

Low and behold the posts are deleted as well.

IMPORTANT!!! The photos for the posts (they exist outside the database) are not deleted.

## Adding images to edit post

Modify the view admin.posts.edit to display the linked photo. See changes in ***bold***.

<div class="row">

***<div class="col-sm-3">***

***<img class="img-responsive" src="{{ $post->photo->file }}" alt="">***

***</div>***

<div ***class="col-sm-9"***>

{!!

Form::model(

$post,

[

'method'=>'PATCH',

'action'=>['AdminPostsController@update', $post->id],

'files'=>true

]

)

!!}

# Section 29: Application - Categories

## Setting up categories

Migration for categories already exist

Model for categories already exists.

Controller for categories needs to be created.

php artisan make:controller --resource AdminCategoriesController

Modify routes.php to add access the CRUD methods for categories. See change in ***bold***.

Route::group(['middleware'=>'admin'], function () {

Route::resource('admin/users', 'AdminUsersController');

Route::resource('admin/posts', 'AdminPostsController');

***Route::resource('admin/categories', 'AdmincategoriesController');***

});

Controller for categories, AdminCategoriesController

public function index()

{

return view('admin.categories.index');

}

. . .

. . .

public function create()

{

return view('admin.categories.create');

}

Modified view layouts.admin. See changes in ***bold***.

<li>

<a href="#">

<i class="fa fa-wrench fa-fw"></i>

Categories

<span class="fa arrow"></span>

</a>

<ul class="nav nav-second-level">

<li>

<a href="***{{route('admin.categories.index')}}***">All Categories</a>

</li>

<li>

<a href="***{{route('admin.categories.create')}}***">Create Category</a>

</li>

</ul>

<!-- /.nav-second-level -->

</li>

Modified view admin.categories.index for testing purposes.

@extends('layouts.admin');

@section('content')

<h1>List Categories</h1>

@endsection

Modified view admin.categories.index for testing purposes.

@extends('layouts.admin');

@section('content')

<h1>Create Categories</h1>

@endsection

Tested All Categories and Create Category, it they work.

## Creating categories

Modified view admin.categories.index. It will now list all categories and provide a category creation window on the same page.

@extends('layouts.admin');

@section('content')

<div class="row">

<h1>Categories</h1>

</div>

<div class="row">

<div class="col-sm-6">

{!!

Form::open([

'method'=>'POST',

'action'=>'AdminCategoriesController@store'

])

!!}

<div class="form-group">

{!! Form::label('name', 'Name:') !!}

{!! Form::text('name', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!! Form::submit('Create Category', ['class'=>'btn btn-primary']) !!}

</div>

{!! Form::close() !!}

</div>

<div class="col-sm-6">

@if($categories)

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Category</th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

@foreach($categories as $category)

<tr>

<td>{{$category->id}}</td>

<td>{{$category->name}}</td>

<td>{{$category->created\_at->diffForHumans()}}</td>

<td>{{$category->updated\_at->diffForHumans()}}</td>

</tr>

@endforeach

</tbody>

</table>

@endif

</div>

</div>

@endsection

Modified the index method in AdminCategoriesController.

public function index()

{

$categories = Category::all();

return view('admin.categories.index', compact('categories'));

}

Modified the store method in AdminCategoriesController.

public function store(Request $request)

{

Category::create($request->all());

return redirect('/admin/categories');

}

For information on Laravel Validation and User Input Sanitization to Prevent XSS Exploits, see: <https://www.cloudways.com/blog/prevent-xss-exploits-using-laravel-validation-and-sanitization/>

## Updating and deleting categories

Modified view admin.categories.index. Turned the category name into a link to the edit/delete page.

<td>

<a href="{{route('admin.categories.edit', $category->id)}}">

{{$category->name}}

</a>

</td>

Modified view admin.categories.edit. Enable the update and delete of a category.

@extends('layouts.admin');

@section('content')

<div class="row">

<h1>Categories</h1>

</div>

<div class="row">

<div class="col-sm-6">

{!!

Form::model(

$category,

[

'method'=>'PATCH',

'action'=>['AdminCategoriesController@update', $category->id]

]

)

!!}

<div class="form-group">

{!! Form::label('name', 'Name:') !!}

{!! Form::text('name', null, ['class'=>'form-control']) !!}

</div>

<div class="form-group">

{!!

Form::submit(

'Update Category',

['class'=>'btn btn-primary col-sm-6']

)

!!}

</div>

{!! Form::close() !!}

{{-- <div class="col-sm-6"> --}}

{!! Form::open([

'method'=>'DELETE',

'action'=>['AdminCategoriesController@destroy', $category->id]

]) !!}

<div class="form-group">

{!!

Form::submit(

'Delete Category',

['class'=>'btn btn-danger col-sm-6']

)

!!}

</div>

{!! Form::close() !!}

{{-- </div> --}}

</div>

</div>

@endsection

Modified the update and destroy methods in AdminCategoriesController.

public function update(Request $request, $id)

{

$category = Category::findOrFail($id);

$category->update($request->all());

return redirect('/admin/categories');

}

public function destroy($id)

{

Category::findOrFail($id)->delete();

return redirect('/admin/categories');

}

No provisions in place to prevent destroying a category that is in use.

Stored everything in git.

git add .

git commit -m 'updated add, edit and delete in categories module'

git push -u origin master

# Section 30: Application - Media

## Setting up and displaying media

Create the view admin.media.index.

@extends('layouts.admin');

@section('content')

<div class="row">

<h1>Media</h1>

</div>

<div class="row">

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>Image</th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

@foreach ($photos as $photo)

<tr>

<td>{!! $photo->id !!}</td>

<td>{!! $photo->file !!}</td>

<td>

<img

height="50" src="{!! $photo->file !!}" alt="Not Available"

>

</td>

<td>

{!!

$photo->created\_at ? $photo->created\_at : 'Not Available'

!!}

</td>

<td>

{!!

$photo->updated\_at ? $photo->updated\_at : 'Not Available'

!!}

</td>

</tr>

@endforeach

</tbody>

</table>

</div>

@endsection

Create a controller for media without the typical resources.

php artisan make:controller AdminMediasController

Modify routes.php to create routes to edit the media table. See changes in ***bold***.

Route::group(['middleware'=>'admin'], function () {

Route::resource('admin/users', 'AdminUsersController');

Route::resource('admin/posts', 'AdminPostsController');

Route::resource('admin/categories', 'AdminCategoriesController');

***Route::resource('admin/media', 'AdminMediasController');***

});

Modify the view layouts.admin to add the new routes to the Media section. See changes in ***bold***.

<li>

<a href="#">

<i class="fa fa-wrench fa-fw"></i>Media<span class="fa arrow"></span>

</a>

<ul class="nav nav-second-level">

<li>

<a href="***{{route('admin.media.index')}}***">All Media</a>

</li>

<li>

<a href="***{{route('admin.media.upload')}}***">Upload Media</a>

</li>

</ul>

<!-- /.nav-second-level -->

</li>

Modify Controller AdminMediasController. See changes in ***bold***.

<?php

namespace App\Http\Controllers;

***use App\Photo;***

use Illuminate\Http\Request;

use App\Http\Requests;

class AdminMediasController extends Controller

{

***public function index()***

***{***

***$photos = Photo::all();***

***return view('admin.media.index', compact('photos'));***

***}***

}

Modify routes.php to provide a route for Media Upload.

Route::group(['middleware'=>'admin'], function () {

Route::resource('admin/users', 'AdminUsersController');

Route::resource('admin/posts', 'AdminPostsController');

Route::resource('admin/categories', 'AdminCategoriesController');

Route::resource('admin/media', 'AdminMediasController');

Route::get(

'admin/media/upload',

['as'=>'admin.media.upload', 'uses'=>'AdminMediasController@store']

);

});

## Setting up the create view

Simplify routes.php by using the create method for the Upload option. Delete the statement:

Route::get(

'admin/media/upload',

['as'=>'admin.media.upload', 'uses'=>'AdminMediasController@store']

);

Modify the view layouts.admin to take advantage of the modification.

<li>

<a href="{{route('admin.media.create')}}">Upload Media</a>

</li>

Modify the controller AdminMediasController to add a create method.

public function create()

{

return view('admin.media.create');

}

Create the view admin.media.create

@extends('layouts.admin');

@section('content')

<div class="row">

<h1>Upload Media</h1>

</div>

@endsection

## Adding the upload plugin

Use Dropzone.js

Purpose: For uploading multiple files.

For more information see: <https://www.dropzonejs.com/>

Location of CDN: <https://cdnjs.com/libraries/dropzone>

Copy the link to the minified JavaScript module:

<https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.5.1/min/dropzone.min.js>

Copy the link to the minified CSS module:

<https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.5.1/min/dropzone.min.css>

Modify the view admin.media.create

At the bottom add the link to the Dropzone JavaScript:

@section('scripts')

<script src="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.5.1/min/

dropzone.min.js">

</script>

@endsection

At the top, after @extends add the link to the Dropzone CSS:

@section('styles')

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/

dropzone/5.5.1/min/dropzone.min.css">

@endsection

Now that @sections have been created in the child view you have to create @yields in the parent view

Modify the view layouts.admin. See changes in ***bold***.

In the <head> section add.

<!-- Bootstrap Core CSS -->

. . .

. . .

. . .

***@yield('styles')***

At the bottom of the view add:

<script src="{{asset('js/libs.js')}}"></script>

@yield('footer')

***@yield('scripts')***

</body>

</html>

## Uploading images

Modify the store method in the controller AdminMediasController

public function store(Request $request)

{

$file = $request->file('file');

$name = time() . $file->getClientOriginalName();

$file->move('images', $name);

Photo::create(['file'=>$name]);

return redirect('/admin/media');

}

With dropzone, once an image lands in dropzone it automatically fires off the store method.

Once you see a momentary checkmark on an image dropped into dropzone the store method has successfully run. The image has been uploaded, renamed, moved to the images directory and its new filename has been stored in the database.

## Deleting images

Modify the destroy method in the controller AdminMediasController

public function destroy($id)

{

$photo = Photo::findOrFail($id);

$photoFileName = strstr($photo->file, '/images/');

unlink(public\_path() . $photoFileName);

$photo->delete();

return redirect('/admin/media');

}

Modify the view admin.media.index. Add the following as the last data column of the table.

<td>

{!!

Form::open([

'method'=>'DELETE',

'action'=>['AdminMediasController@destroy', $photo->id]

])

!!}

<div class="form-group">

{!! Form::submit('Delete', ['class'=>'btn btn-danger']) !!}

</div>

{!! Form::close() !!}

</td>

This adds a delete button at the end of each record.

# Section 31: Application - Comments

## Setting up part 1

Consists of Comments table and Replies table

Under D:\wwwroot\tutorial\codehacking\resources\views\admin create the subdirectory comments

Under D:\wwwroot\tutorial\codehacking\resources\views\admin\comments create the subdirectory replies

Under admin.comments create the view index

@extends('layouts.admin');

@section('content')

<div class="row">

<h1>Comments</h1>

</div>

<div class="row">

</div>

@endsection

Under admin.comments.replies create the view index

@extends('layouts.admin');

@section('content')

<div class="row">

<h1>Replies</h1>

</div>

<div class="row">

</div>

@endsection

Modify routes.php to add the two new route sets. See changes in ***bold***.

Route::group(['middleware'=>'admin'], function () {

Route::resource('admin/users', 'AdminUsersController');

Route::resource('admin/posts', 'AdminPostsController');

Route::resource('admin/categories', 'AdminCategoriesController');

Route::resource('admin/media', 'AdminMediasController');

***Route::resource('admin/comemnts', 'PostCommentsController');***

***Route::resource('admin/comemnt/replies', 'CommentRepliesController');***

});

Create the migration and model for Comments

php artisan make:model Comment -m

Create the migration and model for CommentReplies

php artisan make:model CommentReply -m

Modified migration 2019\_08\_16\_013917\_create\_comments\_table.php

public function up()

{

Schema::create('comments', function (Blueprint $table) {

$table->increments('id');

$table->integer('post\_id')->unsigned()->index();

$table->integer('is\_active')->default(0);

$table->string('author');

$table->string('email');

$table->text('body');

$table->timestamps();

$table->foreign('post\_id')->references('id')->on('posts')->onDelete('cascade');

});

}

Note the foreign key constraint

Modified migration 2019\_08\_16\_013957\_create\_comment\_replies\_table.php

public function up()

{

Schema::create('comment\_replies', function (Blueprint $table) {

$table->increments('id');

$table->integer('comment\_id')->unsigned()->index();

$table->integer('is\_active')->default(0);

$table->string('author');

$table->string('email');

$table->text('body');

$table->timestamps();

$table->foreign('comment\_id')->references('id')->on('comments')->OnDelete('cascade');

});

}

Note the foreign key constraint

Applied migrations.

## Relations and mass assignment

Modified the Post model adding the comments method.

public function comments()

{

return $this->hasMany('App\Comment');

}

Modified the Comment model adding the replies method.

public function replies()

{

return $this->hasMany('App\CommentReply');

}

Modified the CommentReply model adding the comment method.

public function comment()

{

return $this->belongsTo('App\Comment');

}

Create the controller PostCommentsController.

php artisan make:controller --resource PostCommentsController

Create the controller CommentRepliesController.

php artisan make:controller --resource CommentRepliesController

## Setting up views

Modify (fix) routes.php.

Route::resource('admin/comemnts', 'PostCommentsController');

Becomes

Route::resource('admin/***comments***', 'PostCommentsController');

Modify the view layouts.admin. Add an “All Comments” sub-option under Posts.

<li>

<a href="{{route('admin.comments.index')}}">All Comments</a>

</li>

Modify index method of the controller PostCommentsController.

public function index()

{

return "THIS WORKS!!!";

}

Successfully test it.

Modify index method of the controller PostCommentsController. You will now make a call to the view.

public function index()

{

$comments = Comment::all();

return view('admin.comments.index', compact('comments'));

}

A minimalist version of admin.comments.index has already been created a few lecturers earlier.

## Creating a post part 1

Download course-images.zip as Lecture 257 - course-images.zip

Downloaded blog-files.zip as Lecture 257 - blog-files.zip

From Lecture 257 - blog-files.zip opened the file blog-home.html and pasted it to layouts.blog-home.blade.php

From Lecture 257 - blog-files.zip opened the file blog-post.html and pasted it to layouts.blog-post.blade.php

Create a skeleton view named posts for displaying posts.

Modified routes.php. See changes in ***bold***.

***Route::get('/post/{id}', 'AdminPostsController@post');***

Route::group(['middleware'=>'admin'], function () {

***Route::get('/admin', function () {***

***return view('admin.index');***

***});***

. . .

. . .

});

Moved /admin within the protection of the admin middleware.

Create a route for ‘/post’ under the protection of the web middleware (implied)

Create a skeleton method post() in the controller AdminPostsController.

## Creating a post part 2

Modify routes.php. See changes in ***bold***.

Route::get(

'/post/{id}',

['as'=>'home.post', 'uses'=>'AdminPostsController@post']

);

What advantages does this buy you?

You can name your routes, so when you need to for example submit the form to a particular route, instead of typing the route path inside action attribute, like:

<form method="post" action="/post/$post->id">

You can instead specify the route name inside route() helper:

<form method="post" action="{‌{ route('home.post') }}">

Behind the scenes Laravel will match that request with suitable route under that name.

For more information see: <https://laravel.com/docs/5.8/helpers#method-route>

Modify the post method of the controller AdminPostsController.

public function post($id)

{

$post = Post::findOrFail($id);

return view('post', compact('post'));

}

Added the load of a particular post and the redirect to the view post with the post data.

Modified the view layouts.blog-post.

Removed old link to CSS and added links to the bootstrap CSS assets.

<!-- Bootstrap Core CSS -->

<link href="{{asset('css/app.css')}}" rel="stylesheet">

<link href="{{asset('css/libs.css')}}" rel="stylesheet">

Added links to the bootstrap JavaScript.

<!-- jQuery -->

<script src="{{asset('js/libs.js')}}"></script>

Moved the Blog Post section from layouts.blog-post to post.

Modified the view post to receive the Blog Post section from layouts.blog-post.

@extends('layouts.blog-post')

@section('content')

<!-- Blog Post -->

<!-- Title -->

<div class="row">

<h1>{{ $post->title }}</h1>

</div>

<!-- Author -->

<p class="lead">

by <a href="#">Start Bootstrap</a>

</p>

<hr>

<!-- Date/Time -->

<p>

<span class="glyphicon glyphicon-time"></span>

Posted on August 24, 2013 at 9:00 PM

</p>

<hr>

<!-- Preview Image -->

<img class="img-responsive" src="http://placehold.it/900x300" alt="">

<hr>

<!-- Post Content -->

<p class="lead">

Lorem ipsum dolor sit amet, consectetur adipis . . . . .

</p>

<p>

Lorem ipsum dolor sit amet, consectetur adipis . . . . .

</p>

<p>

Lorem ipsum dolor sit amet, consectetur adipis . . . . .

</p>

<p>

Lorem ipsum dolor sit amet, consectetur adipis . . . . .

</p>

<p>

Lorem ipsum dolor sit amet, consectetur adipis . . . . .

</p>

<hr>

<!-- Blog Comments -->

<!-- Comments Form -->

<div class="well">

<h4>Leave a Comment:</h4>

<form role="form">

<div class="form-group">

<textarea class="form-control" rows="3"></textarea>

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</div>

<hr>

<!-- Posted Comments -->

<!-- Comment -->

<div class="media">

<a class="pull-left" href="#">

<img class="media-object" src="http://placehold.it/64x64" alt="">

</a>

<div class="media-body">

<h4 class="media-heading">Start Bootstrap

<small>August 25, 2014 at 9:30 PM</small>

</h4>

Cras sit amet nibh libero, in gravida nulla. . . . . .

</div>

</div>

<!-- Comment -->

<div class="media">

<a class="pull-left" href="#">

<img class="media-object" src="http://placehold.it/64x64" alt="">

</a>

<div class="media-body">

<h4 class="media-heading">Start Bootstrap

<small>August 25, 2014 at 9:30 PM</small>

</h4>

Cras sit amet nibh libero, in gravida nulla. . . . . .

<!-- Nested Comment -->

<div class="media">

<a class="pull-left" href="#">

<img

class="media-object"

src="http://placehold.it/64x64"

alt=""

>

</a>

<div class="media-body">

<h4 class="media-heading">Nested Start Bootstrap

<small>August 25, 2014 at 9:30 PM</small>

</h4>

Cras sit amet nibh libero, in gravida nulla. . . . . .

</div>

</div>

<!-- End Nested Comment -->

</div>

</div>

@endsection

A good portion of this HTML will be replaced by PHP and Blade syntax in the next lecture.

## Creating a post part 3

Modified part of the view post replacing the mock-up HTML with Laravel code.

. . .

. . .

<!-- Author -->

<p class="lead">

by <a href="#">{{ $post->user->name }}</a>

</p>

<hr>

<!-- Date/Time -->

<p>

<span class="glyphicon glyphicon-time"></span>

Posted {{ $post->created\_at->diffForHumans() }}

</p>

<hr>

<!-- Preview Image -->

<img class="img-responsive" src="{{ $post->photo->file }}" alt="">

<hr>

<!-- Post Content -->

<p class="lead">{{ $post->body }}</p>

<hr>

. . .

. . .

Modified the controller AdminPostsController to pass a list of the categories on file. See changes in ***bold***.

public function post($id)

{

$post = Post::findOrFail($id);

***$categories = Category::orderBy('name')->get(['name', 'id']);***

return view('post', compact('post', ***'categories'***));

}

Modified the layout layouts.blog-post to display the list of categories. This is a partial solution. Full solution will be covered later. Hopefully it will cover a two column solution for Laravel v.5.2.

<!-- Blog Categories Well -->

<div class="well">

<h4>Blog Categories</h4>

<div class="row">

<div class="col-lg-6">

<ul class="list-unstyled">

@foreach ($categories as $category)

<li><a href="#">{{ $category->name }}</a></li>

@endforeach

</ul>

</div>

<div class="col-lg-6">

<ul class="list-unstyled">

<li><a href="#">Category Name</a></li>

<li><a href="#">Category Name</a></li>

<li><a href="#">Category Name</a></li>

<li><a href="#">Category Name</a></li>

</ul>

</div>

</div>

<!-- /.row -->

</div>

## Creating a comment

Modify the view post, specifically the section on creating the comment.

<!-- Comments Form -->

<div class="well">

<h4>Leave a Comment:</h4>

{{

Form::open(['method'=>'POST', 'action'=>'PostCommentsController@store'])

}}

{!! Form::hidden('post\_id', $post->id) !!}

<div class="form-group">

{!! Form::label('body', 'Comment:') !!}

{!!

Form::textarea('body', null, ['class'=>'form-control', 'rows'=>3])

!!}

</div>

<div class="form-group">

{!! Form::submit('Submit Comment', ['class'=>'btn btn-primary']) !!}

</div>

{{ Form::close() }}

</div>

Modified the store method of PostCommentsController.

public function store(Request $request)

{

$user = Auth::user();

$photoName = substr(strstr($user->photo->file, '/images/'),8);

$data = [

'post\_id'=>$request->post\_id,

'author'=>$user->name,

'email'=>$user->email,

'photo'=>$photoName,

'body'=>$request->body

];

Comment::create($data);

$request->session()->flash(

'comment\_message',

'Your comment has been submitted and is awaiting moderation.'

);

return redirect()->back();

}

The column photo is created in the migration created later in the lecture.

$user->photo->file will return a filename with a directory prefix because of the accessor in the photo model. That prefix needs to be stripped off so that only the filename gets stored in the photo field. Hence the need for $photoName.

Modified $fillable in the Comment model. See changes in ***bold***.

protected $fillable = [

'post\_id',

'author',

'email',

'body',

***'photo',***

'is\_active'

];

Modified $fillable in the CommentReply model. See changes in ***bold***.

protected $fillable = [

'comment\_id',

'author',

'email',

'body',

***'photo',***

'is\_active'

];

Created migration to add a photo name column to the comments and comment\_replies tables.

Changes to the up method: $table->string('photo');

Changes to the down method: $table->dropColumn('photo');

## Displaying posts

Modified the model Comment to provide a reverse lookup to the post table.

public function post()

{

return $this->belongsTo('App\Post');

}

Made sure the index method in the controller PostCommentsController was passing all the comments to the admin.comments.index view.

Modified the view admin.comments.index to display the comments and if there are no comments display a no comments message.

@extends('layouts.admin');

@section('content')

@if(count($comments) > 0)

<div class="row">

<h1>Comments</h1>

</div>

<div class="row">

<table class="table">

<thead>

<tr>

<th class="text-center">Id</th>

<th class="text-center">Author</th>

<th class="text-center">Email</th>

<th class="text-center">Comment</th>

<th class="text-center">Created</th>

<th class="text-center">Updated</th>

</tr>

</thead>

<tbody>

@foreach($comments as $comment)

<tr>

<td>{{ $comment->id }}</td>

<td>{{ $comment->author }}</td>

<td>{{ $comment->email }}</td>

<td>{{ $comment->body }}</td>

<td>

<a href="{{ route('home.post', $comment->post->id) }}">

View Post

</a>

</td>

<td>{{ $comment->created\_at }}</td>

<td>{{ $comment->updated\_at }}</td>

</tr>

@endforeach

</tbody>

</table>

</div>

@else

<div class="row">

<h1 class="text-center">No Comments</h1>

</div>

@endif

@endsection

## Approving, un-approving and deleting part 1 - setting it up

Modified the view admin.comments.index by adding Approve/Unapprove and delete buttons. The methods update and destroy in the controller PostCommentsController will be updated in the next lecture.

<td>

@if($comment->is\_active == 1)

{!! Form::open([

'method'=>'PATCH',

'action'=>['PostCommentsController@update', $comment->id]

]) !!}

<input type="hidden" name="is\_active" value="0">

<div class="form-group">

{!! Form::submit('Unapprove', ['class'=>'btn btn-success']) !!}

</div>

{!! Form::close() !!}

@else

{!! Form::open([

'method'=>'PATCH',

'action'=>['PostCommentsController@update', $comment->id]

]) !!}

<input type="hidden" name="is\_active" value="1">

<div class="form-group">

{!! Form::submit('Approve', ['class'=>'btn btn-info']) !!}

</div>

{!! Form::close() !!}

@endif

</td>

<td>

{!! Form::open([

'method'=>'DELETE',

'action'=>['PostCommentsController@destroy', $comment->id]

]) !!}

<div class="form-group">

{!! Form::submit('Delete', ['class'=>'btn btn-danger']) !!}

</div>

{!! Form::close() !!}

</td>

## Approving, un-approving and deleting part 2 - Using it

Modified the methods update and destroy in the controller PostCommentsController.

public function update(Request $request, $id)

{

Comment::findOrFail($id)->update($request->all());

return redirect('/admin/comments');

}

The column is\_active is all that is being passed in $request.

public function destroy($id)

{

Comment::findOrFail($id)->delete();

return redirect()->back();

}

## Displaying comments

Modified view admin.comments.index. Added a View Post link.

<td>

<a href="{{ route('home.post', $comment->post->id) }}">

{{ $comment->post->title }}

</a>

</td>

Modified admin.posts.index. Added View posts and View Comments links.

<td>

<a href="{{ route('home.post', $post->id) }}">

View Post

</a>

</td>

<td>

<a href="{{ route('admin.comments.show', $post->id) }}">

View Comments

</a>

</td>

Created admin.comments.show to display the filtered comments. Basically a clone of admin.comments.index with a few less statements. In truth you could have use admin.comments.index instead.

@extends('layouts.admin');

@section('content')

@if(count($comments))

<div class="row">

<h1>Comments</h1>

</div>

<div class="row">

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Author</th>

<th>Email</th>

<th>Comment</th>

<th>Post</th>

<th>Status</th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

@foreach($comments as $comment)

<tr>

<td>{{ $comment->id }}</td>

<td>{{ $comment->author }}</td>

<td>{{ $comment->email }}</td>

<td>{{ $comment->body }}</td>

<td>

<a href="{{ route('home.post', $comment->post->id) }}">

{{ $comment->post->title }}

</a>

</td>

<td>

{!! Form::open([

'method'=>'PATCH',

'action'=>['PostCommentsController@update', $comment->id]

]) !!}

@if($comment->is\_active == 1)

<input type="hidden" name="is\_active" value="0">

<div class="form-group">

{!! Form::submit(

'Unapprove',

['class'=>'btn btn-success']

) !!}

</div>

@else

<input type="hidden" name="is\_active" value="1">

<div class="form-group">

{!! Form::submit('Approve', ['class'=>'btn btn-info']) !!}

</div>

@endif

{!! Form::close() !!}

</td>

<td>

{!! Form::open([

'method'=>'DELETE',

'action'=>['PostCommentsController@destroy', $comment->id]

]) !!}

<div class="form-group">

{!! Form::submit('Delete', ['class'=>'btn btn-danger']) !!}

</div>

{!! Form::close() !!}

</td>

<td>{{ $comment->created\_at }}</td>

<td>{{ $comment->updated\_at }}</td>

</tr>

@endforeach

</tbody>

</table>

</div>

@else

<div class="row">

<h1 class="text-center">No Comments</h1>

</div>

@endif

@endsection

## Displaying specific comments

Modify the show method of the controller PostCommentsController.

public function show($id)

{

$post = Post::findOrFail($id);

$comments = $post->comments;

return view('admin.comments.show', compact('comments'));

}

## Displaying comments in post page

Modified the view post. This will add a check to see if the user is logged in. If they can they can make a comment otherwise no access to the comment field. See changes in ***bold***.

<!-- Blog Comments -->

***@if(Auth::check())***

<!-- Comments Form -->

<div class="well">

<h4>Leave a Comment:</h4>

{{ Form::open([

'method'=>'POST',

'action'=>'PostCommentsController@store'

]) }}

. . .

. . .

{{ Form::close() }}

</div>

***@endif***

Modify post method of AdminPostsController to pass comments that are active to the view posts. See changes in ***bold***.

public function post($id)

{

$post = Post::findOrFail($id);

**$comments = $post->comments()->whereIsActive(1)->get();**

$categories = Category::orderBy('name')->get(['name', 'id']);

return view('post', compact('post', ***'comments',*** 'categories'));

}

Laravel allows for a lot of flexibility. It allows us to write shorthand for the where clause. In this case “where” and the column name (“Is\_Active”) are concatenated and the underscore is dropped.

If we were not using whereIsActive we would need to write the query like this:

$post->comments()->where('is\_active', 1)->get();

Modified the view post. This will display the comments, if there are any and that have been flagged as active.

<!-- Posted Comments -->

@if(count($comments) > 0)

<!-- Comment -->

@foreach($comments as $comment)

<div class="media">

<a class="pull-left" href="#">

<img

height="64"

class="media-object"

src="/codehacking/public/images/{{ $comment->photo }}"

alt=""

>

</a>

<div class="media-body">

<h4 class="media-heading">{{ $comment->author }}

<small>

Created: {{ $comment->created\_at->diffForHumans() }}

Last Updated: {{ $comment->updated\_at->diffForHumans() }}

</small>

</h4>

{{ $comment->body }}

</div>

</div>

@endforeach

@endif

## Creating the reply form

Modify the view post to create the reply form.

<!-- Nested Comment -->

. . .

. . .

{{ Form::open([

'method'=>'POST',

'action'=>'CommentRepliesController@createReply'

]) }}

<div class="form-group">

{!! Form::label('body', 'Reply:') !!}

{!! Form::textarea(

'body',

null,

['class'=>'form-control', 'rows'=>1]

) !!}

</div>

<div class="form-group">

{!! Form::submit('Create Reply', ['class'=>'btn btn-primary']) !!}

</div>

{{ Form::close() }}

</div>

<!-- End Nested Comment -->

Modify routes.php to create the POST route to CommentRepliesController

Route::group(['middleware'=>'auth'], function () {

Route::post('comment/reply', 'CommentRepliesController@createReply');

});

“Protected” by the auth middleware. Meaning that regular users can access it and you don’t have to be an admin.

Modify the createReply method of CommentRepliesController. Just to see if connectivity works.

public function createReply(Request $request)

{

return 'IT WORKS!!!!';

}

## Creating a reply

Enter and save a reply to a comment.

Modify the view post. A hidden field containing the comment id needs to be added. See changes in ***bold***.

{{ Form::open(

['method'=>'POST', 'action'=>'CommentRepliesController@createReply']

) }}

***{!! Form::hidden('comment\_id', $comment->id) !!}***

. . .

. . .

{{ Form::close() }}

Modify the createReply method of the controller CommentRepliesController.

public function createReply(Request $request)

{

$user = Auth::user();

$photoName = substr(strstr($user->photo->file, '/images/'), 8);

$data = [

'comment\_id'=>$request->comment\_id,

'author'=>$user->name,

'email'=>$user->email,

'photo'=>$photoName,

// 'photo'=>$user->photo->file,

'body'=>$request->body

];

CommentReply::create($data);

$request->session()->flash('reply\_message', 'Your reply has been submitted’

. ' and is awaiting moderation.');

return redirect()->back();

}

Tested it, it works. And now for the display.

## Displaying replies

Modified the view post to show the replies under the comments.

@if(count($comment->replies) > 0)

@foreach($comment->replies as $reply)

<!-- Nested Comment -->

<div id="nested-comment" class="media">

<a class="pull-left" href="#">

<img

height="64"

class="media-object"

src="/codehacking/public/images/{{ $reply->photo }}" alt=""

>

</a>

<div class="media-body">

<h4 class="media-heading">{{ $reply->author }}

<small>

Created: {{ $reply->created\_at->diffForHumans() }}

Last Updated: {{ $reply->updated\_at->diffForHumans() }}

</small>

</h4>

{{ $reply->body }}

</div>

{{ Form::open([

'method'=>'POST',

'action'=>'CommentRepliesController@createReply'

]) }}

{!! Form::hidden('comment\_id', $comment->id) !!}

<div class="form-group">

{!! Form::label('body', 'Reply:') !!}

{!! Form::textarea(

'body',

null,

['class'=>'form-control', 'rows'=>1]

) !!}

</div>

<div class="form-group">

{!! Form::submit('Create Reply', ['class'=>'btn btn-primary']) !!}

</div>

{{ Form::close() }}

</div>

<!-- End Nested Comment -->

@endforeach

@endif

Modified the css file {application root}\resources\assets\sass\app.scss. Added a paragraph to increase vertical spacing between replies and comments.

#nested-comment {

margin-top: 60px;

}

Ran gulp to incorporate the changes made to app.css.

PS D:\wwwroot\tutorial\codehacking> ***gulp***

[13:20:47] Using gulpfile D:\wwwroot\tutorial\codehacking\gulpfile.js

[13:20:47] Starting 'default'...

[13:20:47] Starting 'sass'...

Fetching Sass Source Files...

- resources\assets\sass\app.scss

Saving To...

- public\css\app.css

[13:20:48] Finished 'default' after 1.63 s

[13:20:48] gulp-notify: [Laravel Elixir] Sass Compiled!

[13:20:48] Finished 'sass' after 1.73 s

[13:20:48] Starting 'styles'...

Fetching Styles Source Files...

- resources\assets\css\libs\blog-post.css

- resources\assets\css\libs\bootstrap.css

- resources\assets\css\libs\font-awesome.css

- resources\assets\css\libs\metisMenu.css

- resources\assets\css\libs\sb-admin-2.css

Saving To...

- ./public/css/libs.css

[13:20:48] gulp-notify: [Laravel Elixir] Stylesheets Merged!

[13:20:48] Finished 'styles' after 81 ms

[13:20:48] Starting 'scripts'...

Fetching Scripts Source Files...

- resources\assets\js\libs\jquery.js

- resources\assets\js\libs\bootstrap.js

- resources\assets\js\libs\metisMenu.js

- resources\assets\js\libs\sb-admin-2.js

- resources\assets\js\libs\scripts.js

Saving To...

- ./public/js/libs.js

[13:20:49] gulp-notify: [Laravel Elixir] Scripts Merged!

[13:20:49] Finished 'scripts' after 809 ms

PS D:\wwwroot\tutorial\codehacking>

Had to clear the cache for the changes in the css files to be reflected in the web pages.

In the end it all worked.

## Displaying replies with JavaScript

Add a reply button that will hide and reveal the reply form.

JavaScript will be used to accomplish this, specifically jQuery.

Modify the view layouts.blog-post. See changes in ***bold***.

<!-- jQuery -->

<script src="{{asset('js/libs.js')}}"></script>

***@yield('scripts')***

Modify the css file {application root}\resources\assets\sass\app.scss. Added a paragraph to hide the reply form.

.comment-reply {

display: none;

}

Ran gulp to incorporate the changes made to app.css.

Modify the view post to add div’s, classes and JavaScript (jQuery) to hide and show the reply form. See changes in ***bold***.

<!-- Comment -->

@foreach($comments as $comment)

<div class="media">

<a class="pull-left" href="#">

<img height="64" class="media-object"

src="/codehacking/public/images/{{ $comment->photo }}" alt=""

>

</a>

<div class="media-body">

<h4 class="media-heading">{{ $comment->author }}

<small>

Created: {{ $comment->created\_at->diffForHumans() }}

Last Updated: {{ $comment->updated\_at->diffForHumans() }}

</small>

</h4>

<p>{{ $comment->body }}</p>

@if(count($comment->replies) > 0)

@foreach($comment->replies as $reply)

<!-- Nested Comment -->

<div id="nested-comment" class="media">

<a class="pull-left" href="#">

<img height="64" class="media-object"

src="/codehacking/public/images/{{ $reply->photo }}"

alt=""

>

</a>

<div class="media-body">

<h4 class="media-heading">{{ $reply->author }}

<small>

Created: {{ $reply->created\_at->diffForHumans() }}

Last Updated: {{ $reply->updated\_at->diffForHumans() }}

</small>

</h4>

{{ $reply->body }}

</div>

***<div class="comment-reply-container">***

***<button class="toggle-reply btn btn-rpimary pull-right">***

***Reply***

***</button>***

***<div class="comment-reply">***

{{ Form::open([

'method'=>'POST',

'action'=>'CommentRepliesController@createReply'

]) }}

{!! Form::hidden('comment\_id', $comment->id) !!}

<div class="form-group">

{!! Form::label('body', 'Reply:') !!}

{!! Form::textarea(

'body', null, ['class'=>'form-control', 'rows'=>1]

) !!}

</div>

<div class="form-group">

{!! Form::submit(

'Create Reply', ['class'=>'btn btn-primary']

) !!}

</div>

{{ Form::close() }}

***</div>***

***</div>***

</div>

<!-- End Nested Comment -->

@endforeach

@endif

</div>

</div>

@endforeach

@endif

@endsection

***@section('scripts')***

***<script>***

***$(".comment-reply-container .toggle-reply").click(function() {***

***$(this).next().slideToggle("slow");***

***});***

***</script>***

***@endsection***

Tested it. It works.

## Displaying replies in admin part 1

Fix the slide down appearance of the reply form. Now it does not slide down and inflate side to side. It only slides down.

<div class="comment-reply col-sm-9">

Created the view admin.comments.replies.show by cloning admin.comments.show and making the appropriate changes.

@extends('layouts.admin');

@section('content')

@if(count($replies))

<div class="row">

<h1>Replies</h1>

</div>

<div class="row">

<table class="table">

<thead>

<tr>

<th>Id</th>

<th>Author</th>

<th>Email</th>

<th>Comment</th>

<th>Post</th>

<th>Status</th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

@foreach($replies as $reply)

<tr>

<td>{{ $reply->id }}</td>

<td>{{ $reply->author }}</td>

<td>{{ $reply->email }}</td>

<td>{{ $reply->body }}</td>

<td>

<a href="{{ route('home.post', $reply->comment->post->id) }}">

{{ $reply->comment->post->title }}

</a>

</td>

<td>

{!! Form::open([

'method'=>'PATCH',

'action'=>['CommentRepliesController@update', $reply->id]

]) !!}

@if($reply->is\_active == 1)

<input type="hidden" name="is\_active" value="0">

<div class="form-group">

{!! Form::submit(

'Unapprove', ['class'=>'btn btn-success']

) !!}

</div>

@else

<input type="hidden" name="is\_active" value="1">

<div class="form-group">

{!! Form::submit(

'Approve',

['class'=>'btn btn-info']

) !!}

</div>

@endif

{!! Form::close() !!}

</td>

<td>

{!! Form::open([

'method'=>'DELETE',

'action'=>['CommentRepliesController@destroy', $reply->id]

]) !!}

<div class="form-group">

{!! Form::submit('Delete', ['class'=>'btn btn-danger']) !!}

</div>

{!! Form::close() !!}

</td>

<td>{{ $reply->created\_at }}</td>

<td>{{ $reply->updated\_at }}</td>

</tr>

@endforeach

</tbody>

</table>

</div>

@else

<div class="row">

<h1 class="text-center">No Replies</h1>

</div>

@endif

@endsection

## Displaying replies in admin part 2

Modified admin.comments.index to add a column to link from the comment to the associated replies.

<td>

<a href="{{ route('admin.comment.replies.show', $comment->id) }}">

View Replies

</a>

</td>

Modified the show method of the controller CommentRepliesController.

public function show($id)

{

$comment = Comment::findOrFail($id);

$replies = $comment->replies;

return view('admin.comments.replies.show', compact('replies'));

}

Tested it. The replies come up. Now to work on the Approve/Unapproved and Delete buttons. On to the next lecture.

## Updating and deleting replies

Modify the view post to only show active replies. See changes in ***bold***.

@if(count($comment->replies) > 0)

@foreach($comment->replies as $reply)

***@if($reply->is\_active == 1)***

<!-- Nested Comment -->

<div id="nested-comment" class="media">

<a class="pull-left" href="#">

<img

height="64"

class="media-object"

src="/codehacking/public/images/{{ $reply->photo }}"

alt=""

>

</a>

<div class="media-body">

<h4 class="media-heading">{{ $reply->author }}

<small>

Created: {{ $reply->created\_at->diffForHumans() }}

Last Updated: {{ $reply->updated\_at->diffForHumans() }}

</small>

</h4>

{{ $reply->body }}

</div>

<div class="comment-reply-container">

<button class="toggle-reply btn btn-rpimary pull-right">

Reply

</button>

<div class="comment-reply col-sm-9">

{{ Form::open([

'method'=>'POST',

'action'=>'CommentRepliesController@createReply'

]) }}

{!! Form::hidden('comment\_id', $comment->id) !!}

<div class="form-group">

{!! Form::label('body', 'Reply:') !!}

{!! Form::textarea(

'body',

null,

['class'=>'form-control', 'rows'=>1]

) !!}

</div>

<div class="form-group">

{!! Form::submit(

'Create Reply',

['class'=>'btn btn-primary']

) !!}

</div>

{{ Form::close() }}

</div>

</div>

</div>

<!-- End Nested Comment -->

***@endif***

@endforeach

***@else***

***<h1 class="text-center">No Replies</h1>***

@endif

Modify the view admin.comments.replies.show, change the column title comment to reply to reduce confusion.

<th>Reply</th>

Modify update method of the controller CommentRepliesController

public function update(Request $request, $id)

{

CommentReply::findOrFail($id)->update($request->all());

return redirect()->back();

}

Modify destroy method of the controller CommentRepliesController

public function destroy($id)

{

CommentReply::findOrFail($id)->delete();

return redirect()->back();

}

# Section 32: Extra Features

## Pretty url configuration part 1

Too many issues. Did not try again until the end of the Section 33. Upgrade to Laravel 5.3. See lecture 289. Upgrading Our Pretty URL Feature.

## Pretty url configuration and finished part 2 - *Revisit after upgrading to laravel v.5.3*

## Laravel Pagination

Pagination – showing x number of record on the screen at a time instead of all the records.

Modify the index method of AdminPostsController.

public function index()

{

// $posts = Post::All(); //original non-paginated statement

$posts = Post::paginate(2); //create pages of two records

return view('admin.posts.index', compact('posts'));

}

Modify admin.posts.index to take advantage of the pagination. See changes in ***bold***.

</table>

<div class="row">

<div class="col-sm-6 col-sm-offset-5">

{{ $posts->render() }}

</div>

</div>

This will display the page number bar along with forward and backward arrows.

For more information see: <https://laravel.com/docs/5.1/pagination>

## Pulling Gravatars from users

Gravatar – Globally recognized avatar: <https://en.gravatar.com/>

How to use Gravatars: <https://wordpress.org/support/article/how-to-use-gravatars/>

Size of the image you give to Gravatar should be no bigger than 300 x 300 pixels.

Quick summary: generate a MD5 hash of an email address and use that hash to request the gravatar for that email address.

Now to get it to work.

Modify user model.

public function getGravatarAttribute()

{

$hash = md5(strtolower(trim($this->attributes['email'])));

return "http://www.gravatar.com/avatar/$hash";

}

This will create a new “column” or attribute in the model but not in the database. That attribute is called Gravater. It’s an md5 has of the users email address.

Modify the view post.

Original statement:

<img height="64" class="media-object"

src="/codehacking/public/images/{{ $comment->photo }}"

alt="">

Modified statement:

<img height="64" class="media-object"

src="{{ Auth::user()->gravatar }}"

alt="">

When Laravel sees the call for the attribute it will automatically call the accessor getGravatarAttribute.

Tested. It works.

# Section 33: Upgrade to Laravel 5.3

## Brief introduction to changes

For more information see: <https://laravel-news.com/look-whats-coming-laravel-5-3>

Some highlights;

Laravel Mailable

Laravel Notifications

Laravel Passport

Rollback Migrations

Expanded Blade Foreach Loops with the $loop variable

Additional information: <https://scotch.io/tutorials/a-deep-dive-into-laravel-5-3-features>

Laravel Deployment: <https://www.udemy.com/course/laravel-deployment/#overview>

## Laravel project reference files download - version 5.3

Updated project files: 5.3\_laravel\_project\_files.zip

Renamed to : lecture 279 - 5.3\_laravel\_project\_files.zip

## Upgrading - Extra folders in this version

Changes from 5.2 to 5.3

Routes.php moved:

From: {application root}\app\Http

To: {application root}\routes

Routes.php has been split into three files:

Api.php

Console.php

Web.php (contains welcome banner)

App folder contains the minimal number of folders. Folders like events, listeners abd policies do not get created until they are actually needed. (php artisan make:…..) Means smaller footprint.

Edit composer.json changing the version number for laravel/framework and laravelcollective/framework to 5.3.\* from 5.2.\*

"laravel/framework": "5.3.\*",

"laravelcollective/html": "5.3.\*",

Save changes to composer.json and run composer update

Errors due to changes in Laravel:

Running composer update will generate the following error message:

[ErrorException]

Declaration of App\Providers\EventServiceProvider::boot(Illuminate\Contracts\ Events\Dispatcher $events) should be compatible with Illuminate\ Foundation\Support\Providers\EventServiceProvider::boot()

To resolve this error go to the boot method of {application root}/app/Providers/ EventServiceProvider.php and remove the $events parameter

Before:

/\*\*

\* Register any other events for your application.

\*

***\* @param \Illuminate\Contracts\Events\Dispatcher $events***

\* @return void

\*/

***public function boot(DispatcherContract $events)***

{

***parent::boot($events);***

}

After:

/\*\*

\* Register any other events for your application.

\*

***\* @param***

\* @return void

\*/

***public function boot()***

{

***parent::boot();***

}

Next attempt at running composer update, next error:

[ErrorException]

Declaration of App\Providers\RouteServiceProvider::boot(Illuminate\Routing\Router $router) should be compatible with Illuminate\Foundation\Support\ Providers\RouteServiceProvider::boot()

To resolve this error go to the boot, map and mapWebRoutes method of {application root}/app/Providers/RouteServiceProvider.php and remove the $router parameter.

Before:

/\*\*

\* Define your route model bindings, pattern filters, etc.

\*

***\* @param \Illuminate\Routing\Router $router***

\* @return void

\*/

***public function boot(Router $router)***

{

***parent::boot($router);***

}

/\*\*

\* Define the routes for the application.

\*

***\* @param \Illuminate\Routing\Router $router***

\* @return void

\*/

***public function map(Router $router)***

{

***$this->mapWebRoutes($router);***

}

/\*\*

\* Define the "web" routes for the application.

\*

\* These routes all receive session state, CSRF protection, etc.

\*

***\* @param \Illuminate\Routing\Router $router***

\* @return void

\*/

***protected function mapWebRoutes(Router $router)***

{

***$router->group([***

'namespace' => $this->namespace, 'middleware' => 'web',

], function ($router) {

require app\_path('Http/routes.php');

});

}

After:

/\*\*

\* Define your route model bindings, pattern filters, etc.

\*

***\* @param***

\* @return void

\*/

***public function boot()***

{

***parent::boot();***

}

/\*\*

\* Define the routes for the application.

\*

***\* @param***

\* @return void

\*/

***public function map()***

{

***$this->mapWebRoutes();***

}

/\*\*

\* Define the "web" routes for the application.

\*

\* These routes all receive session state, CSRF protection, etc.

\*

***\* @param***

\* @return void

\*/

***protected function mapWebRoutes()***

{

**Route::group([**

'namespace' => $this->namespace, 'middleware' => 'web',

], function ($router) {

require app\_path('Http/routes.php');

});

}

In the method mapWebRoutes, the method group has to be called statically from Route.

Once the above editing has been done add the following statement to the top to resolve for Route.

use Illuminate\Support\Facades\Route;

And delete the statement:

use Illuminate\Routing\Router;

In the method mapWebRoutes modify the statement:

require app\_path('Http/routes.php');

and change it to:

require base\_path(***'routes/web.php'***);

Create the directory routes under {application route} and extract the three files from the download class archive Lecture 280 - routes.zip and place them in the routes directory.

Next attempt at running composer update.

It looks like it worked!!!!

PS D:\wwwroot\tutorial\codehacking> composer update

Loading composer repositories with package information

Updating dependencies (including require-dev)

Nothing to install or update

Package phpunit/phpunit-mock-objects is abandoned, you should avoid using it. No replacement was suggested.

Generating autoload files

Carbon 1 is deprecated, see how to migrate to Carbon 2.

https://carbon.nesbot.com/docs/#api-carbon-2

You can run ".\vendor\bin\upgrade-carbon" to get help in updating carbon and other frameworks and libraries that depend on it.

> Illuminate\Foundation\ComposerScripts::postUpdate

> php artisan optimize

Generating optimized class loader

The compiled class file has been removed.

Double check the version of Laravel

PS D:\wwwroot\tutorial\codehacking> php artisan –version

Laravel Framework version 5.3.31

When checking the route list you end up with:

PS D:\wwwroot\tutorial\codehacking> php artisan route:list

+--------+----------+-----+------+---------+------------+

| Domain | Method | URI | Name | Action | Middleware |

+--------+----------+-----+------+---------+------------+

| | GET|HEAD | / | | Closure | web |

+--------+----------+-----+------+---------+------------+

Now to upgrade the rest of the routes.

## Upgrading routes part 1

Copy the routes encapsulated in Route::group(['middleware'=>'admin'] . . . from:

{application root}\app\Http\routes.php

To:

{application root}\routes\web.php

End state of web.php:

<?php

/\*

|--------------------------------------------------------------------------

| Web Routes

|--------------------------------------------------------------------------

|

| This file is where you may define all of the routes that are handled

| by your application. Just tell Laravel the URIs it should respond

| to using a Closure or controller method. Build something great!

|

\*/

Auth::routes();

Route::get('/', function () {

return view('welcome');

});

Route::get('/home', 'HomeController@index');

Route::get('/post/{id}', ['as'=>'home.post', 'uses'=>'AdminPostsController@post']);

Route::group(['middleware'=>'admin'], function () {

Route::get('/admin', function () {

return view('admin.index');

});

Route::resource('admin/users', 'AdminUsersController');

Route::resource('admin/posts', 'AdminPostsController');

Route::resource('admin/categories', 'AdminCategoriesController');

Route::resource('admin/media', 'AdminMediasController');

Route::resource('admin/comments', 'PostCommentsController');

Route::resource('admin/comment/replies', 'CommentRepliesController');

});

/\*

| Give access to a regular user and not an admin

\*/

Route::group(['middleware'=>'auth'], function () {

Route::post('comment/reply', 'CommentRepliesController@createReply');

});

Now we try to get a list of the routes and we get this error:

PS D:\wwwroot\tutorial\codehacking> php artisan route:list

[Symfony\Component\Debug\Exception\FatalErrorException]

Trait 'Illuminate\Foundation\Auth\Access\AuthorizesResources' not found

This forces the modification of the file {application root}\app\Http\Controllers\Controller.php

The statement:

use AuthorizesRequests, ***AuthorizesResources,*** DispatchesJobs, ValidatesRequests;

Changes to:

use AuthorizesRequests, DispatchesJobs, ValidatesRequests;

And it works . . . . Almost.

PS D:\wwwroot\tutorial\codehacking> php artisan route:list

+------+---------+---------------------+------------------+------------------------------------------------------+----------+

|Domain|Method |URI |Name |Action |Middleware|

+------+---------+---------------------+------------------+------------------------------------------------------+----------+

| |GET|HEAD |/ | |Closure |web |

| |GET|HEAD |admin | |Closure |web,admin |

| |GET|HEAD |admin/categories |categories.index |App\Http\Controllers\AdminCategoriesController@index |web,admin |

| |POST |admin/categories |categories.store |App\Http\Controllers\AdminCategoriesController@store |web,admin |

| |GET|HEAD |admin/categories |categories.create |App\Http\Controllers\AdminCategoriesController@create |web,admin |

| | | /create | | | |

| |PUT|PATCH|admin/categories |categories.update |App\Http\Controllers\AdminCategoriesController@update |web,admin |

| | | /{category} | | | |

| |GET|HEAD |admin/categories |categories.show |App\Http\Controllers\AdminCategoriesController@show |web,admin |

| | | /{category} | | | |

| |DELETE |admin/categories |categories.destroy|App\Http\Controllers\AdminCategoriesController@destroy|web,admin |

| | | /{category} | | | |

| |GET|HEAD |admin/categories |categories.edit |App\Http\Controllers\AdminCategoriesController@edit |web,admin |

| | |/{category}/edit | | | |

| |POST |admin/comment |replies.store |App\Http\Controllers\CommentRepliesController@store |web,admin |

| | | /replies | | | |

| |GET|HEAD |admin/comment |replies.index |App\Http\Controllers\CommentRepliesController@index |web,admin |

| | | /replies | | | |

| |GET|HEAD |admin/comment |replies.create |App\Http\Controllers\CommentRepliesController@create |web,admin |

| | | /replies/create | | | |

| |DELETE |admin/comment |replies.destroy |App\Http\Controllers\CommentRepliesController@destroy |web,admin |

| | |/replies/{reply} | | | |

| |PUT|PATCH|admin/comment |replies.update |App\Http\Controllers\CommentRepliesController@update |web,admin |

| | |/replies/{reply} | | | |

| |GET|HEAD |admin/comment |replies.show |App\Http\Controllers\CommentRepliesController@show |web,admin |

| | |/replies/{reply} | | | |

| |GET|HEAD |admin/comment |replies.edit |App\Http\Controllers\CommentRepliesController@edit |web,admin |

| | |/replies/{reply}/edit| | | |

| |POST |admin/comments |comments.store |App\Http\Controllers\PostCommentsController@store |web,admin |

| |GET|HEAD |admin/comments |comments.index |App\Http\Controllers\PostCommentsController@index |web,admin |

| |GET|HEAD |admin/comments/create|comments.create |App\Http\Controllers\PostCommentsController@create |web,admin |

| |GET|HEAD |admin/comments |comments.show |App\Http\Controllers\PostCommentsController@show |web,admin |

| | | /{comment}| | | |

| |DELETE |admin/comments |comments.destroy |App\Http\Controllers\PostCommentsController@destroy |web,admin |

| | | /{comment}| | | |

| |PUT|PATCH|admin/comments |comments.update |App\Http\Controllers\PostCommentsController@update |web,admin |

| | | /{comment}| | | |

| |GET|HEAD |admin/comments/ |comments.edit |App\Http\Controllers\PostCommentsController@edit |web,admin |

| | | {comment}/edit| | | |

| |GET|HEAD |admin/media |media.index |App\Http\Controllers\AdminMediasController@index |web,admin |

| |POST |admin/media |media.store |App\Http\Controllers\AdminMediasController@store |web,admin |

| |GET|HEAD |admin/media/create |media.create |App\Http\Controllers\AdminMediasController@create |web,admin |

| |DELETE |admin/media/{medium} |media.destroy |App\Http\Controllers\AdminMediasController@destroy |web,admin |

| |GET|HEAD |admin/media/{medium} |media.show |App\Http\Controllers\AdminMediasController@show |web,admin |

| |PUT|PATCH|admin/media/{medium} |media.update |App\Http\Controllers\AdminMediasController@update |web,admin |

| |GET|HEAD |admin/media/{medium} |media.edit |App\Http\Controllers\AdminMediasController@edit |web,admin |

| | | /edit| | | |

| |GET|HEAD |admin/posts |posts.index |App\Http\Controllers\AdminPostsController@index |web,admin |

| |POST |admin/posts |posts.store |App\Http\Controllers\AdminPostsController@store |web,admin |

| |GET|HEAD |admin/posts/create |posts.create |App\Http\Controllers\AdminPostsController@create |web,admin |

| |DELETE |admin/posts/{post} |posts.destroy |App\Http\Controllers\AdminPostsController@destroy |web,admin |

| |PUT|PATCH|admin/posts/{post} |posts.update |App\Http\Controllers\AdminPostsController@update |web,admin |

| |GET|HEAD |admin/posts/{post} |posts.show |App\Http\Controllers\AdminPostsController@show |web,admin |

| |GET|HEAD |admin/posts/{post} |posts.edit |App\Http\Controllers\AdminPostsController@edit |web,admin |

| | | /edit| | | |

| |GET|HEAD |admin/users |users.index |App\Http\Controllers\AdminUsersController@index |web,admin |

| |POST |admin/users |users.store |App\Http\Controllers\AdminUsersController@store |web,admin |

| |GET|HEAD |admin/users/create |users.create |App\Http\Controllers\AdminUsersController@create |web,admin |

| |GET|HEAD |admin/users/{user} |users.show |App\Http\Controllers\AdminUsersController@show |web,admin |

| |PUT|PATCH|admin/users/{user} |users.update |App\Http\Controllers\AdminUsersController@update |web,admin |

| |DELETE |admin/users/{user} |users.destroy |App\Http\Controllers\AdminUsersController@destroy |web,admin |

| |GET|HEAD |admin/users/{user} |users.edit |App\Http\Controllers\AdminUsersController@edit |web,admin |

| | | /edit| | | |

+------+---------+---------------------+------------------+------------------------------------------------------+----------+

In the name column the “admin.” prefix is missing. That means anywhere the function route(‘admin. . . .’) is used it will fail. This will be fixed in lecture “Upgrading routes part 3 - Route names”.

The next error to fix is a problem with accessing the login screen.

## Upgrading routes part 2 - Login

Fixing login screen access.

First thing to do is go to the Laravel 5.3 upgrade documentation. Specifically the section titled “Authentication Scaffolding” (see: <https://laravel.com/docs/5.3/upgrade#upgrade-5.3.0>)

It explains the following:

The two default authentication controllers provided with the framework have been split into four smaller controllers. This change provides cleaner, more focused authentication controllers by default. The easiest way to upgrade your application to the new authentication controllers is to grab a fresh copy of each controller from GitHub and place them into your application.

Follow the link: <https://github.com/laravel/laravel/tree/5.3/app/Http/Controllers/Auth>

Copy and paste the four files:

ForgotPasswordController.php

LoginController.php

RegisterController.php

ResetPasswordController.php

If the login screen does not appear, make sure the statement below appears in the file {application root}/routes/web.php

Auth::routes();

Login and Logout are now working.

## Upgrading routes part 3 - Route names

Modify {application root}/routes/web.php in order to give the routes the necessary names for the application to work.

Route::resource('admin/users', 'AdminUsersController'***, ['names'=>[***

***'index'=>'admin.users.index',***

***'create'=>'admin.users.create',***

***'store'=>'admin.users.store',***

***'edit'=>'admin.users.edit'***

***]]***);

Route::resource('admin/posts', 'AdminPostsController'***, ['names'=>[***

***'index'=>'admin.posts.index',***

***'create'=>'admin.posts.create',***

***'store'=>'admin.posts.store',***

***'edit'=>'admin.posts.edit'***

***]]***);

Route::resource('admin/categories', 'AdminCategoriesController'***, ['names'=>[***

***'index'=>'admin.categories.index',***

***'create'=>'admin.categories.create',***

***'store'=>'admin.categories.store',***

***'edit'=>'admin.categories.edit'***

***]]***);

Route::resource('admin/media', 'AdminMediasController'***, ['names'=>[***

***'index'=>'admin.media.index',***

***'create'=>'admin.media.create',***

***'store'=>'admin.media.store',***

***'edit'=>'admin.media.edit'***

***]]***);

Route::resource('admin/comments', 'PostCommentsController'***, ['names'=>[***

***'index'=>'admin.comments.index',***

***'create'=>'admin.comments.create',***

***'store'=>'admin.comments.store',***

***'edit'=>'admin.comments.edit'***

***]]***);

Route::resource('admin/comment/replies', 'CommentRepliesController'***,***

***['names'=>[***

***'index'=>'admin.comment.replies.index',***

***'create'=>'admin.comment.replies.create',***

***'store'=>'admin.comment.replies.store',***

***'edit'=>'admin.comment.replies.edit'***

***]***

***]***);

When you run php artisan route:list you get the following:

+------+---------+-------------------------+---------------------+--------------------------------------------------+----------+

|Domain|Method |URI |Name |Action |Middleware|

+------+---------+-------------------------+---------------------+--------------------------------------------------+----------+

| |GET|HEAD |/ | |Closure |web |

| |GET|HEAD |admin | |Closure |web,admin |

***| |GET|HEAD |admin/categories |admin.categories |App\Http\Controllers\AdminCategoriesController |web,admin |***

***| | | | .index | @index | |***

***| |POST |admin/categories |admin.categories |App\Http\Controllers\AdminCategoriesController |web,admin |***

***| | | | .store | @store | |***

***| |GET|HEAD |admin/categories/create |admin.categories |App\Http\Controllers\AdminCategoriesController |web,admin |***

***| | | | .create | @create| |***

| |DELETE |admin/categories |categories.destroy |App\Http\Controllers\AdminCategoriesController |web,admin |

| | | /{category} | | @destroy| |

| |PUT|PATCH|admin/categories |categories.update |App\Http\Controllers\AdminCategoriesController |web,admin |

| | | /{category} | | @update| |

| |GET|HEAD |admin/categories |categories.show |App\Http\Controllers\AdminCategoriesController |web,admin |

| | | /{category} | | @show| |

***| |GET|HEAD |admin/categories |admin.categories |App\Http\Controllers\AdminCategoriesController |web,admin |***

***| | | /{category}/edit | .edit | @edit| |***

***| |POST |admin/comment/replies |admin.comment.replies|App\Http\Controllers\CommentRepliesController |web,admin |***

***| | | | .store| @store| |***

***| |GET|HEAD |admin/comment/replies |admin.comment.replies|App\Http\Controllers\CommentRepliesController |web,admin |***

***| | | | .index| @index| |***

***| |GET|HEAD |admin/comment/replies |admin.comment.replies|App\Http\Controllers\CommentRepliesController |web,admin |***

***| | | /create | .create| @create| |***

| |GET|HEAD |admin/comment/replies/ |replies.show |App\Http\Controllers\CommentRepliesController@show|web,admin |

| | | {reply} | | | |

| |PUT|PATCH|admin/comment/replies |replies.update |App\Http\Controllers\CommentRepliesController |web,admin |

| | | /{reply} | | @update | |

| |DELETE |admin/comment/replies |replies.destroy |App\Http\Controllers\CommentRepliesController |web,admin |

| | | /{reply} | | @destroy | |

***| |GET|HEAD |admin/comment/replies |admin.comment |App\Http\Controllers\CommentRepliesController@edit|web,admin |***

***| | | /{reply}/edit | .replies.edit| | |***

***| |POST |admin/comments |admin.comments.store |App\Http\Controllers\PostCommentsController@store |web,admin |***

***| |GET|HEAD |admin/comments |admin.comments.index |App\Http\Controllers\PostCommentsController@index |web,admin |***

***| |GET|HEAD |admin/comments/create |admin.comments.create|App\Http\Controllers\PostCommentsController@create|web,admin |***

| |GET|HEAD |admin/comments/{comment} |comments.show |App\Http\Controllers\PostCommentsController@show |web,admin |

| |DELETE |admin/comments/{comment} |comments.destroy |App\Http\Controllers\PostCommentsController |web,admin |

| | | | | @destroy| |

| |PUT|PATCH|admin/comments/{comment} |comments.update |App\Http\Controllers\PostCommentsController@update|web,admin |

***| |GET|HEAD |admin/comments/{comment} |admin.comments.edit |App\Http\Controllers\PostCommentsController@edit |web,admin |***

***| | | /edit | | | |***

***| |POST |admin/media |admin.media.store |App\Http\Controllers\AdminMediasController@store |web,admin |***

***| |GET|HEAD |admin/media |admin.media.index |App\Http\Controllers\AdminMediasController@index |web,admin |***

***| |GET|HEAD |admin/media/create |admin.media.create |App\Http\Controllers\AdminMediasController@create |web,admin |***

| |GET|HEAD |admin/media/{medium} |media.show |App\Http\Controllers\AdminMediasController@show |web,admin |

| |PUT|PATCH|admin/media/{medium} |media.update |App\Http\Controllers\AdminMediasController@update |web,admin |

| |DELETE |admin/media/{medium} |media.destroy |App\Http\Controllers\AdminMediasController@destroy|web,admin |

***| |GET|HEAD |admin/media/{medium}/edit|admin.media.edit |App\Http\Controllers\AdminMediasController@edit |web,admin |***

***| |POST |admin/posts |admin.posts.store |App\Http\Controllers\AdminPostsController@store |web,admin |***

***| |GET|HEAD |admin/posts |admin.posts.index |App\Http\Controllers\AdminPostsController@index |web,admin |***

***| |GET|HEAD |admin/posts/create |admin.posts.create |App\Http\Controllers\AdminPostsController@create |web,admin |***

| |DELETE |admin/posts/{post} |posts.destroy |App\Http\Controllers\AdminPostsController@destroy |web,admin |

| |PUT|PATCH|admin/posts/{post} |posts.update |App\Http\Controllers\AdminPostsController@update |web,admin |

| |GET|HEAD |admin/posts/{post} |posts.show |App\Http\Controllers\AdminPostsController@show |web,admin |

***| |GET|HEAD |admin/posts/{post}/edit |admin.posts.edit |App\Http\Controllers\AdminPostsController@edit |web,admin |***

***| |POST |admin/users |admin.users.store |App\Http\Controllers\AdminUsersController@store |web,admin |***

***| |GET|HEAD |admin/users |admin.users.index |App\Http\Controllers\AdminUsersController@index |web,admin |***

***| |GET|HEAD |admin/users/create |admin.users.create |App\Http\Controllers\AdminUsersController@create |web,admin |***

| |DELETE |admin/users/{user} |users.destroy |App\Http\Controllers\AdminUsersController@destroy |web,admin |

| |GET|HEAD |admin/users/{user} |users.show |App\Http\Controllers\AdminUsersController@show |web,admin |

| |PUT|PATCH|admin/users/{user} |users.update |App\Http\Controllers\AdminUsersController@update |web,admin |

***| |GET|HEAD |admin/users/{user}/edit |admin.users.edit |App\Http\Controllers\AdminUsersController@edit |web,admin |***

| |POST |comment/reply | |App\Http\Controllers\CommentRepliesController |web,auth |

| | | | | @createReply | |

| |GET|HEAD |home | |App\Http\Controllers\HomeController@index |web,auth |

| |POST |login | |App\Http\Controllers\Auth\LoginController@login |web,guest |

| |GET|HEAD |login |login |App\Http\Controllers\Auth\LoginController |web,guest |

| | | | | @showLoginForm | |

| |GET|HEAD |logout | |App\Http\Controllers\Auth\LoginController@logout |web |

| |POST |logout |logout |App\Http\Controllers\Auth\LoginController@logout |web |

| |POST |password/email | |App\Http\Controllers\Auth\ForgotPasswordController|web,guest |

| | | | | @sendResetLinkEmail| |

| |POST |password/reset | |App\Http\Controllers\Auth\ResetPasswordController |web,guest |

| | | | | @reset | |

| |GET|HEAD |password/reset | |App\Http\Controllers\Auth\ForgotPasswordController|web,guest |

| | | | | @showLinkRequestForm| |

| |GET|HEAD |password/reset/{token} | |App\Http\Controllers\Auth\ResetPasswordController |web,guest |

| | | | | @showResetForm | |

| |GET|HEAD |post/{id} |home.post |App\Http\Controllers\AdminPostsController@post |web |

| |POST |register | |App\Http\Controllers\Auth\RegisterController |web,guest |

| | | | | @register | |

| |GET|HEAD |register |register |App\Http\Controllers\Auth\RegisterController |web,guest |

| | | | | @showRegistrationForm | |

+------+---------+-------------------------+---------------------+--------------------------------------------------+----------+

Some but not all of the routes have been updated. More testing is needed to find all the necessary routes.

## Upgrading users

In the upgrade the method lists() has been deprecated. Use pluck() in its place. See changes in ***bold***.

This means modifying the create and edit methods in the controller AdminUsersController.

public function create()

{

$roles = Role::***pluck***('name', 'id')->all();

return view('admin.users.create', compact('roles'));

}

public function edit($id)

{

$user = User::findOrFail($id);

$roles = Role::***pluck***('name', 'id')->all();

return view('admin.users.edit', compact('user', 'roles'));

}

## Upgrading posts

Migrate the route:

Route::get(

'/post/{id}',

['as'=>'home.post', 'uses'=>'AdminPostsController@post']

);

From {application root}\app\http\routes.php to {application root}\routes\web.php.

In {application root}\routes\web.php add the named route for show to the PostCommentsController. See changes in ***bold***.

Route::resource('admin/comments', 'PostCommentsController', ['names'=>[

'index'=>'admin.comments.index',

'create'=>'admin.comments.create',

'store'=>'admin.comments.store',

'edit'=>'admin.comments.edit',

***'show'=>'admin.comments.show'***

]]);

And it means modifying the create and edit methods in the controller AdminPostsController.

public function create()

{

$categories = Category::***pluck***('name', 'id')->all();

return view('admin.posts.create', compact('categories'));

}

public function edit($id)

{

$post = Post::findOrFail($id);

$categories = Category::***pluck***('name', 'id')->all();

return view('admin.posts.edit', compact('post', 'categories'));

}

The version (v.3.1 )of Eloquent-Sluggable used is section 32 lectures 274 and 275 on pretty URL’s is not compatible with the Laravel 5.3 upgrade. It needs to be upgraded as well. Have not implemented it yet. Not a problem.

## Upgrading categories

Only issue is that a route does not exist for create categories. It wasn’t necessary as everything is taken care of on a single page.

Solution: Go to the view layouts.admin and either remove or comment out the option to “Create Category”

## Checking out media

All good. No issues with the Media section.

## Upgrading middleware

For more information see: <https://laravel.com/docs/5.3/upgrade#upgrade-5.3.0>

Modify {application root}\app\Http \Kernel.php.

In the $routeMiddleware property the class can has been upgraded from:

'can' => \Illuminate\Foundation\Http\Middleware\Authorize::class,

To:

'can' => \Illuminate\Auth\Middleware\Authorize::class,

Route model binding is now accomplished using middleware. All applications should add the following to the web middleware group ($middlewareGroups = ['web' => []]):

\Illuminate\Routing\Middleware\SubstituteBindings::class,

You should also register a route middleware for binding substitution in the $routeMiddleware property:

'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,

Modify the ‘api’ group of $middlewareGroups with the following:

'api' => [

'throttle:60,1',

'bindings',

],

## Upgrading our Pretty URL feature

Installing the package Eloquent-Sluggable.

On GitHub search for cviebrock/eloquent-sluggable

On Packagist search for cviebrock/eloquent-sluggable

To do an automated install with composer use the following (grabs latest version that fits your environment):

composer require cviebrock/eloquent-sluggable

To force the specific version that is in use by the course at the moment do the following:

Modify composer.json and try to upgrade "cviebrock/eloquent-sluggable" from “^3.1” to "^4.0". If you already have a version installed yoou probably just have to change the version number. If you do not have a prior version installed insert the whole line. See changes in ***bold***.

"require": {

"php": ">=5.5.9",

"laravel/framework": "5.3.\*",

"laravelcollective/html": "5.3.\*",

***"cviebrock/eloquent-sluggable": "^4.0",***

"filp/whoops": "^2.5"

},

Run composer update. If there are errors go to {application root}\config\app.php and remove any references to Sluggable under the providers group this may take care of the problem. Try rerunning composer update.

To configure Sluggable

Modify {application root}\config\app.php and add the following change in ***bold***.

'providers' => [

. . .

. . .

Collective\Html\HtmlServiceProvider::class,

***Cviebrock\EloquentSluggable\ServiceProvider::class,***

/\*

. . .

. . .

Run the command below to place the configuration file for Sluggable on the {application root}\config directory.

php artisan vendor:publish --provider="Cviebrock\EloquentSluggable\ServiceProvider"

Upon successful execution modify the file {application root}\config\sluggable.php. Make the change in ***bold***.

'onUpdate' => ***true***,

Modify the model Post and make the changes in ***bold***.

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

***use Cviebrock\EloquentSluggable\Sluggable;***

***use Cviebrock\EloquentSluggable\SluggableScopeHelpers;***

class Post extends Model

{

***use Sluggable;***

***use SluggableScopeHelpers;***

protected $fillable = [

'category\_id',

'photo\_id',

'title',

'body',

***'slug'***

];

***/\*\****

***\* Return the sluggable configuration array for this model***

***\****

***\* @return array***

***\*/***

***public function sluggable()***

***{***

***return [***

***'slug' => [***

***'source' => 'title'***

***]***

***];***

***}***

. . .

. . .

}

Created the migration to create the slug column in the posts table. With the command:

php artisan make:migration add\_slug\_to\_posts\_table --table=”posts”

Edit the migration. See database migration 2019\_08\_25\_155338\_add\_slug\_to\_posts\_table.php.

Run the migration to update the Posts table.

You should be at a point where saving or creating a post updates the slug field.

Create new posts or update existing posts in order to load the slug field.

Modify the view admin.posts.index to send the slug instead of the post’s id. See change in ***bold***.

<td><a href="{{ route('home.post', $post->***slug***) }}">View Post</a></td>

Modify the post method of AdminPostsController

public function post($slug)

{

$post = Post::findBySlugOrFail($slug);

$comments = $post->comments()->whereIsActive(1)->get();

$categories = Category::orderBy('name')->get(['name', 'id']);

return view('post', compact('post', 'comments', 'categories'));

}

Done. Check the address line. You should see a modified title in place of the id.

For more information see: <https://github.com/cviebrock/eloquent-sluggable>

# Section 34: Extra feature – WYSIWYG and File Management installation

## Part 1- Downloading packages and installing editor

WYSIWYG Editor:

Create the view includes.tinyeditor.

<script src="//cdn.tinymce.com/4/tinymce.min.js"></script>

<script>

tinymce.init(

{

selector:'textarea'

}

);

</script>

Modify the view admin.posts.create inserting the line below after the statement @section('content')

@include('includestinyeditor')

Test "Create Post”. The text area no has formatting options.

Make same changes to the view admin.posts.edit and test.

File Manager:

Before running these installs you need to make sure the php extension php\_fileinfo.dll is installed in php’s ext directory and that it is enabled in php.ini

To determine the php.in file you are using run the command: php --ini

To enable the extension in php.ini add the following to the [ExtensionList] section.

extension=php\_fileinfo.dll

Install the unisharp/laravel-filemanager and intervention/image.

Package #1: unisharp/laravel-filemanager

Install with: composer require unisharp/laravel-filemanager

Package #2: intervention/image

Install with: composer require intervention/image

Modify {application root}\config\app.php and add the following to the end of the ‘providers’ property after the application service providers section.

UniSharp\LaravelFilemanager\LaravelFilemanagerServiceProvider::class,

Intervention\Image\ImageServiceProvider::class,

Keep modifying {application root}\config\app.php and add the following to the end of the ‘aliases’ property.

## 'Image' => Intervention\Image\Facades\Image::class,

Run the following:

php artisan vendor:publish --tag=lfm\_config

To create the file {application root}\config\lfm.php

php artisan vendor:publish --tag=lfm\_public

To create the directory {application root}\public\vendor\laravel-filemanager and its contents.

For more information see:

<https://github.com/UniSharp/laravel-filemanager/blob/master/docs/installation.md>

## Part 2- Making folders and uploading files

You can edit the file {application root}\config\lfm.php to play with the file and graphics options.

By default Laravel File Manager use a different directory from images to store media. This can be changed with the following:

Original setting:

'images\_folder\_name' => 'photos',

New setting:

'images\_folder\_name' => 'images',

Problem with my setup.

The website is not homed to localhost. My URLs are:

http://tutorial.tst/codehacking/public/admin

Instead of:

http://tutorial.tst/admin

This forces a modification to the view tinyeditor. The line:

path\_absolute : "/",

Changes to:

path\_absolute : "/codehacking/public/",

path\_absolute has to contain the folders that lie between localhost and your site.

For more information see: <https://github.com/UniSharp/laravel-filemanager/issues/506>

## Displaying images

Go to the Posts table and modify the body of the Posts that reference an image in the photos folder. It needs to point to the images folder. See changes in ***bold***.

Before:

<img

src="/codehacking/public***/photos/***shares

/Annotation 2019-08-29 182415.jpg"

alt="Idiot in orange"

/>

After:

<img

src="/codehacking/public***/images/***shares

/Annotation 2019-08-29 182415.jpg"

alt="Idiot in orange"

/>

Modifications to the view post and admin.post.edit.

Fix in the event a Post has no photo. This has nothing to do with the body not having an embedded photo. See changes in ***bold***.

Before:

<!-- Preview Image -->

<img

class="img-responsive"

src="***{{ $post->photo->file }}***"

alt=""

>

After:

<!-- Preview Image -->

<img

class="img-responsive"

src="***{{ $post->photo ? $post->photo->file : null }}***"

alt=""

>

To substitute for the null in the above statements we would like to display a placeholder photo. To do that we modify the post model. And add the photoPlaceholder method.

public function photoPlaceholder()

{

return "http://placehold.it/200x200";

}

This also means the modifications made in the views post and admin.post.edit get changed once again.

Before:

src="{{ $post->photo ? $post->photo->file : null }}"

After:

src="{{

$post->photo ? $post->photo->file :

***$post->photoPlaceholder()***

}}"

Modification to display a photo embedded in the body field. Switch from {{ }} to {!! !!}. See changes in ***bold***.

Before:

<p class="lead">{***{*** $post->body ***}***}</p>

After:

<p class="lead">{***!!*** $post->body ***!!***}</p>

# Section 35: Extra feature – Disqus Comment System

## Installing Disqus

Discus is the internet's favorite comment plug-in. It makes it easy for publishers to bring their content to life, fuel colorful discussions, efficiently moderate comments, and build engaged communities.

Website supplying services <https://disqus.com/>

Lecture shows how to integrate the Disqus comment system in place of the programmed comment system in the view post.

# Section 36: Extra feature - Bulk Media Delete

## Part 1 - Creating form, method and route

Doing the prep work.

Create a custom route in web.php.

Route::post('/delete/media', 'AdminMediasController@deleteMedia');

Modify the controller AdminMediasController to be in sync with the route in web.php.

public function deleteMedia(Request $request)

{

dd($request);

}

Modify the view admin.media.index to be in sync with the route in web.php and the controller AdminMediasController. See changes in ***bold***.

<div class="row">

***@if($photos)***

***<form action="/delete/media" method="post" class="form-inline">***

***<div class="form-group">***

***<select name="checkBoxArray" id="" class="form-control">***

***<option value="delete">Delete</option>***

***</select>***

***</div>***

***<div class="form-group">***

***<input type="submit" class="btn btn-primary">***

***</div>***

<table class="table">

<thead>

<tr>

***<th><input type="checkbox" id="options"></th>***

<th>Id</th>

<th>Name</th>

<th>Image</th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

@foreach ($photos as $photo)

<tr>

***<td>***

***<input***

***class="checkBoxes"***

***type="checkbox"***

***name="checkBoxArray[]"***

***value="{{ $photo->id }}"***

***>***

***</td>***

<td>{!! $photo->id !!}</td>

. . .

. . .

</tr>

@endforeach

</tbody>

</table>

***</form>***

***@else***

***<div class="row">***

***<h1 class="text-center">No Media On File</h1>***

***</div>***

***@endif***

</div>

## Part 2 - Testing our form and deleting

Modify a custom route in web.php.

Route::delete('/delete/media', 'AdminMediasController@deleteMedia');

Modify (complete) the deleteMedia() method in the controller AdminMediasController to be in sync with the route in web.php.

public function deleteMedia(Request $request)

{

$photos = Photo::findOrFail($request->checkBoxArray);

foreach ($photos as $photo) {

$photo->delete();

}

return redirect()->back();

}

findOrFail() can have a single id or an array of ids passed to.

Note this only delete the photos from the photo table and the images directory.

This does not go after the photos embedded in the posts or comments.

Modify the view admin.media.index to be in sync with the route in web.php and the controller AdminMediasController. See changes in ***bold***.

@if($photos)

***{{-- href="{{ url('/password/reset') }}" --}}***

***{{--***

***<form action="/codehacking/public/delete/media" method="post" class="form-inline">***

***--}}***

***<form action="{{ url('/delete/media') }}" method="post" class="form-inline">***

***{{ csrf\_field() }}***

***{{ method\_field('delete') }}***

<div class="form-group">

<select name="checkBoxArray" id="" class="form-control">

. . .

. . .

## Part 3 - Adding Javascript / jQuery

Added jQuery to check and uncheck the boxes in the bulk delete column.

@section('scripts')

<script>

$(document).ready(function() {

$('#options').click(function() {

if (this.checked) {

$('.checkBoxes').each(function() {

this.checked = true;

});

} else {

$('.checkBoxes').each(function(){

this.checked = false;

});

}

});

});

</script>

@endsection

#options – The id of the master checkbox.

.click – refers to the click event of the master checkbox being checked and unchecked.

.checkBoxes – the class of each checkbox in the data portion of the table (data row).

.each – Effectively the same as a foreach loop in JavaScript/jQuery

this – Inside the each loop it’s the checkbox currently being processed.

In the if statement it refers to the mast checkbox

## Fixing bulk delete bug and new improvements

This lecture covered how to get the delete checkbox and the delete button to work in a single form and not a form within a form.

Modify the routes in web.php

Before:

Route::delete('delete/media', 'AdminMediasController@deleteMedia');

After:

Route::delete('admin/delete/media', 'AdminMediasController@deleteMedia');

Modify the view admin.media.index

Before:

<form

action="/codehacking/public/delete/media"

method="post"

class="form-inline"

>

After:

<form

action="{{ url('/admin/delete/media') }}"

method="post"

class="form-inline"

>

Before:

<option value="">Delete</option>

After:

<option value="">Delete</option>

Before:

<input type="submit" name="delete\_all" class="btn btn-primary">

After:

<input type="submit" name="delete\_all" class="btn btn-primary">

Before:

<td>

{!! Form::open([

'method'=>'DELETE',

'action'=>['AdminMediasController@destroy', $photo->id]

]) !!}

<input type="hidden" name="photo" value="{{ $photo->id }}">

<div class="form-group">

{!! Form::submit('Delete', ['class'=>'btn btn-danger']) !!}

</div>

{!! Form::close() !!}

</td>

After:

<td>

<div class="form-group">

<input

type="submit"

name="delete\_single[{{ $photo->id }}]"

value="Delete"

class="btn btn-danger"

>

</div>

</td>

Modify the deleteMedia method of the controller AdminMediaController.

public function deleteMedia(Request $request)

{

if (isset($request->delete\_single)) {

$photoId = array\_search('Delete', $request->delete\_single);

$this->destroy($photoId);

}

if (isset($request->delete\_all) && !empty($request->checkBoxArray)) {

$photos = Photo::findOrFail($request->checkBoxArray);

foreach ($photos as $photo) {

$this->destroy($photo->id);

}

}

return redirect()->back();

}

## Removing extra delete button

Goes through the steps to delete the button for deleting a single record.

Modify the view admin.media.index to get rid of the button.

Modify the deleteMedia method of the controller AdminMediaController to remove the logic for deleting a single record.

No I did not do it. I like the additional button.

# Section 37: Extra feature: Deploying our app to a shared hosting account

## Intro

Learn how to install a Laravel application in a shared hosting environment.

Recommend using Go Daddy. Sales pitch.

## SSH - WINDOWS OS

Download and install the latest version of PuTTY on Windows.

How to configure and login with PuTTY.

## SSH - MAC OS

Demonstrate connecting to a webhost with SSH from a Mac.

## Module installation and PHP version

Connect to cPanel

With Go Daddy it’s {domain name}/cpanel

Click on Select PHP Version of PHP and make sure you’re using the latest version and make sure the appropriate extensions and options are checked off.

## Composer install

SSH into the webhost and run:

wget <https://getcomposer.org/installer>

Alternate command for downloading composer

curl –sS <https://getcomposer.org/installer> | php

To check in the downloaded file is good:

php installer --check

If all is good run:

php installer

To get the composer splash & help screens run:

php composer.phar

If all is good you now install Laravel by running:

php composer.phar create-project –prefer-dist laravel/laravel {project name/directory}

Research for additional information and proper format.

Make a bin folder off of root (home)

Move composer.phar to the bin directory.

Make sure the bin directory is in your path

See the composer docs for more info at <https://getcomposer.org/doc/00-intro.md>

## Laravel installer

To install the Laravel Installer run:

composer global require “laravel/installer”

This should create the directory $HOME/.composer/vendor/bin. This directory needs to be added to your $PATH.

Edit .bash\_profile to update your $PATH

To create a new Laravel project all you have to do is run:

laravel new {project name}

For more information see <https://laravel.com/docs/5.4>

## Uploading project

Use cPanel or ssh to move files around on the remote host.

Go to GIT and download a .zip file of your project to your workstation.

Use cPanel to upload the mote host.zip file.

If the remote host has GIT installed you download the file directly to the remote host. In place of the previous two steps.

While in cPanel’s File Manager right click on the .zip file and extract it. You should now see a project directory. In this case codehacking.

You will be missing the .env file, vendor directories and the databases. Those will be created in the next lecture.

## Configuring our environment file

cd ~ should take you to the home (root) directory of the remote host.

Go to the codehacking directory.

Run composer update to install all the packages.

Careful with composer. Composer install will install specified versions in the composer.json file while

composer update will install latest version of the packages specified in the composer.json file.

Once composer is finished php artisan should now work.

Create .env file.

Copy .env.example to .env and edit it.

Set values for DB\_DATABASE, DB\_USERNAME, DB\_PASSWORD.

APP\_KEY will be generated.

Update APP\_URL with the domain and directory where public will reside (<http://domain/directory>).

When the application goes live in production APP\_ENV will be set to production and APP\_DEBUG will be set to false.

Run php artisan key:generate to set APP\_KEY in the .env file.

## Our app live on the web

Move the contents of the public folder to its final destination in this case ~/public\_html/laravel. Make sure all files including the hidden ones like .htaccess are copied as well.

Go to the directory ~/public\_html/laravel and edit the file index.php.

Make changes to reflect the shift in the location of the public directory:

Before: require \_\_DIR\_\_.'/../bootstrap/autoload.php';

After: require \_\_DIR\_\_.'/../../codehacking/bootstrap/autoload.php';

Before: $app = require\_once \_\_DIR\_\_.'/../bootstrap/app.php';

After: $app = require\_once \_\_DIR\_\_.'/../../codehacking/bootstrap/app.php';

Go to the directory ~/codehacking/config and edit the file app.php.

Before: 'url' => env('APP\_URL', 'http://localhost'),

After: 'url' => env('APP\_URL', 'http://{New Domain Name}/{directory where pubic is now located}'),

Change reflects the new domain name for your web site and the new location of the public directory.

## Loging in

Go through the cPanel wizard for creating a MySQL database. Make sure the database name coincides with the one in the .env file.

Same goes for the user id you create for this database. Make sure the name coincides with the one in the .env file. Don’t forget to give the id the proper privileges.

Same goes for the password you assign to the user id you create for this database. Make sure the name coincides with the one in the .env file.

To create the tables in the database run:

php artisan migrate

If the above command fails it’s probably due to the .env file being out of sync with what you created.

Start your application and register the first user.

If you run into 404 Page Missing errors double check that you copied the .htaccess file. As it is hidden it may be missed by a wildcard copy.

Once the first id is created use phpMyAdmin to update the following;

Roles Table

Create Id: 1 Name: administrator

Create Id: 2 Name: author

Create Id: 3 Name: subscriber

User Table (Update first user)

Role\_Id: 1

Is\_Active: 1

The application should now be ready for user login. Test it. There is an outstanding issue with images.

## Displaying images

Already ran into this problem because as my URL is not homed to the public directory.

Question: To get around the pathing issue with the images folder can you create a Super Global to contain the path with a key=value pair (ex.: APP\_IMAGES\_FOLDER=/public\_html/laravel/images) in the .env file and of course the .env.example file? I am assuming this key=value pair would create a Super Global that could be used wherever you need the path to the images directory. Would it be that simple or is there additional coding necessary to create the Super Global?

Answer: Yes you could do that. And then you would access the data with the help of config() helper: config(app.images.folder)

For more information see: <https://laravel.com/docs/5.8/configuration#accessing-configuration-values>

And: <https://laravel-news.com/creating-configuration-in-laravel>

## Deleting images

Problem is unique to instructor’s setup on his remote.

## Setting up production settings

Modify .env file

Before:

APP\_ENV=local

APP\_DEBUG=true

After:

APP\_ENV=production

APP\_DEBUG=false

Changing APP\_DEBUG=false suppresses error messages that would normally be generated.

# Section 38: Upgrade to Laravel 5.4

## Intro and part 1

What version of Laravel are we using:

>php artisan --version

Laravel Framework version 5.3.31

Upgrade guide (Upgrading To 5.4.0 From 5.3):

<https://laravel.com/docs/5.4/upgrade>

Modify the file composer.json with the following:

laravel/framework dependency to 5.4.\*

laravelcollective/html to 5.4.\*

cviebrock/eloquent-sluggable to 4.2

phpunit/phpunit dependency to ~5.7

If it exists, you may delete the bootstrap/cache/compiled.php file.

Once the edits have been saved run:

composer update

Upon successful completion of composer update check the version. It should be 5.4.x:

>php artisan --version

Laravel Framework 5.4.36

Flush the view and route caches:

>php artisan view:clear

Compiled views cleared!

>php artisan route:clear

Route cache cleared!

Test application by creating a post. It should work.

## Laravel upgrade to 5.4 - part two

Download file: webpack-and-packagejson.zip

Rename {application root}\gulpfile.js to {application root}\webpack.mis.js

Reformat the contents of webpack.js to look like the following;

const { mix } = require('laravel-mix');

mix.js('resources/assets/js/app.js', 'public/js')

.sass('resources/assets/sass/app.scss', 'public/css');

mix.styles([

'resources/assets/css/libs/blog-post.css',

'resources/assets/css/libs/bootstrap.css',

'resources/assets/css/libs/font-awesome.css',

'resources/assets/css/libs/metisMenu.css',

'resources/assets/css/libs/sb-admin-2.css'

], 'public/css/libs.css');

mix.scripts([

'resources/assets/js/libs/jquery.js',

'resources/assets/js/libs/bootstrap.js',

'resources/assets/js/libs/sb-admin-2.js',

'resources/assets/js/libs/metisMenu.js',

'resources/assets/js/libs/scripts.js'

], 'public/js/libs.js');

mix.styles contains the .css files that will be processed and outputted to public/css/libs.css as in the gulp processing.

mix.scripts contains the .js files that will be processed and outputted to public/css/libs.js as in the gulp processing.

Delete the directory {application root}\node\_modules. It is no longer needed.

Modify package.json as you are no longer using gulp.

{

"private": true,

"scripts": {

"dev": "node node\_modules/cross-env/dist/bin/cross-env.js NODE\_ENV=development node\_modules/webpack/bin/webpack.js --progress --hide-modules --config=node\_modules/laravel-mix/setup/webpack.config.js",

"watch": "node node\_modules/cross-env/dist/bin/cross-env.js NODE\_ENV=development node\_modules/webpack/bin/webpack.js --watch --progress --hide-modules --config=node\_modules/laravel-mix/setup/webpack.config.js",

"watch-poll": "node node\_modules/cross-env/dist/bin/cross-env.js NODE\_ENV=development node\_modules/webpack/bin/webpack.js --watch --watch-poll --progress --hide-modules --config=node\_modules/laravel-mix/setup/webpack.config.js",

"hot": "node node\_modules/cross-env/dist/bin/cross-env.js NODE\_ENV=development node\_modules/webpack-dev-server/bin/webpack-dev-server.js --inline --hot --config=node\_modules/laravel-mix/setup/webpack.config.js",

"production": "node node\_modules/cross-env/dist/bin/cross-env.js NODE\_ENV=production node\_modules/webpack/bin/webpack.js --progress --hide-modules --config=node\_modules/laravel-mix/setup/webpack.config.js"

},

"devDependencies": {

"axios": "^0.15.3",

"bootstrap-sass": "^3.3.7",

"jquery": "^3.1.1",

"laravel-mix": "^0.8.1",

"lodash": "^4.17.4",

"vue": "^2.2.2"

}

}

## Laravel upgrade to 5.4 - part three

Download the file: appjs-bootstrap-and-components.zip

Extract files to: {application root}\resources\assets\js

Run the following to recreate the node\_modules directory:

npm install

Modified the view layouts.admin.

Before:

<link href="{{asset('css/app.css')}}" rel="stylesheet">

<link href="{{asset('css/libs.css')}}" rel="stylesheet">

After:

<link href="{{asset('css/libs.css')}}" rel="stylesheet">

<link href="{{asset('css/app.css')}}" rel="stylesheet">

Modified the file {application root}\resources\assets\sass\app.scss by adding the following at the bottom (Makes the background black):

body {

background: #000;

}

Run the following to resolve a missing dependency:

npm install --save-dev node-sass@4.12

Run the following to compile the change to app.scss:

npm run dev

Clear the browser cache

Refresh/reload the website you should see the background as black.

## Webpack error - fix

Reviewed a number of issues that have already been resolved by the time it got to me.

The one outstanding issue is in {application root}\webpack.mix.js

Before:

'resources/assets/js/sb-admin-2.js',

After:

'resources/assets/js/***libs/***sb-admin-2.js',

# Section 39: Extra feature: Laravel data seeding

## Creating a simple seeder

How to create dummy data and seed a table with it.

Parent executable of the seed routine is:

{application root}\database\seeds\DatabaseSeeder.php

This calls all other seed files.

Create a seed class:

php artisan make:seeder UserTableSeeder

php artisan make:seeder CategoryTableSeeder

Naming convention:

User and Category – Table name

Table – What is being seeded

Seeder – This is a seeder class

The files/classes CategoryTableSeeder.php and UserTableSeeder.php will be created in the folder {application root}\database\seeds

Modify UserTableSeeder.php to add a single user.

public function run()

{

DB::table('users')->insert([

'name'=>str\_random(10),

'role\_id'=>2,

'is\_active'=>1,

'email'=>str\_random(10) . '@' . 'bsrresearch.com',

'password'=>bcrypt('secret')

]);

}

Modify CategoryTableSeeder.php to add a multiple categories.

public function run()

{

DB::table('categories')->insert(['name'=>'COBOL']);

DB::table('categories')->insert(['name'=>'Algol']);

DB::table('categories')->insert(['name'=>'Fortran']);

DB::table('categories')->insert(['name'=>'BASIC']);

DB::table('categories')->insert(['name'=>'Go']);

DB::table('categories')->insert(['name'=>'Ruby']);

}

Modify DatabaseSeeder.php to run both seed classes.

public function run()

{

$this->call(UsersTableSeeder::class);

$this->call(CategoryTableSeeder::class);

}

To run the seed classes run the following:

php artisan db:seed

You should see the data in the tables.

## Creating a more advanced seeder with factories

Using a factory to create seed data.

Factory is located in {application root}\database\factories\ModelFactory.php

Template for the factory to create users is in ModelFactory.php

$factory->define(App\User::class, function (Faker\Generator $faker) {

return [

'name' => $faker->name,

'email' => $faker->safeEmail,

'password' => bcrypt(str\_random(10)),

'remember\_token' => str\_random(10),

];

});

Factory object calling the previously defined method define

Define has two paramethers:

Model – App\User::class

A function that returns an array of fields that would be inserted into the database.

Here the name and email will contain unique human likeable data instead of a stream of random characters.

password gets an encrypted stream of random characters.

remember\_token will get a string of random characters.

These assignments are on a per record basis. No duplicates between records.

To use the factory we have to go back to the DatabaseSeeder.php and insert the statement:

factory(App\User::class, 10)->create();

You are invoking the factory for App\User 10 times to create records.

You can create related records by chaining factories with the following:

factory(App\User::class, 10)->create()->each(function ($user) {

$user->posts()->save(factory(App\Post::class)->make());

});

For each user record you create invoke the factory (passing the user\_id) for App\Post and have the Post record created with the user record the App\User factory just created.

You have to go back to ModelFactory.php and create the factory for App\Post or this will not work.

## Let's create factories for all our database tables

Modify {application root}\database\factories\ModelFactory.php to create a factory for each table.

Factory for Users table.

$factory->define(App\User::class, function (Faker\Generator $faker) {

return [

'name' => $faker->name,

'email' => $faker->safeEmail,

'password' => bcrypt(str\_random(10)),

'role\_id' => $faker->numberBetween(1, 3),

'remember\_token' => str\_random(10),

];

});

Factory for Posts table.

$factory->define(App\Post::class, function (Faker\Generator $faker) {

return [

'category\_id' => $faker->numberBetween(1, 10),

'photo\_id' => 1,

'title' => $faker->sentence(7, 11),

'body' => $faker->paragraphs(rand(10, 15), true),

'slug' => $faker->slug()

];

});

Factory for Roles table.

$factory->define(App\Role::class, function (Faker\Generator $faker) {

return [

'name' => $faker->unique()->randomElement(['Administrator', 'Author',

'Subscriber']),

];

});

Use of unique() ensures there are no duplicate role names. If DatabaseSeeder.php tries to create more records than there are elements it will cause an error.

Factory for Categories table.

$factory->define(App\Category::class, function (Faker\Generator $faker) {

return [

'name' => $faker->unique()->randomElement(['PHP', 'Programming',

'JavaScript', 'Life', 'Travel','Coffee', 'Money', 'Women',

'Men', 'Love' ]),

];

});

Use of unique() ensures there are no duplicate category names. If DatabaseSeeder.php tries to create more records than there are elements it will cause an error.

Factory for Photos table.

$factory->define(App\Photo::class, function (Faker\Generator $faker) {

return [ 'file' => 'placeholder.jpg' ];

});

Factory for Comments table.

$factory->define(App\Comment::class, function (Faker\Generator $faker) {

return [

'post\_id' => $faker->numberBetween(1, 10),

'is\_active' => 1,

'author' => $faker->name,

'photo' => 'placeholder.jpg',

'email' => $faker->safeEmail,

'body' => $faker->paragraphs(1, true),

];

});

Factory for Comment\_Replies table.

$factory->define(App\CommentReply::class, function (Faker\Generator $faker) {

return [

'is\_active'=> 1,

'author'=> $faker->name,

'photo'=> 'placeholder.jpg',

'email' => $faker->safeEmail,

'body' => $faker->paragraphs(1, true),

];

});

Modify {application root}\database\factories\DatabaseSeeder.php to actually put the factories to use.

Temporarily disable foreign key validation

DB::statement('SET FOREIGN\_KEY\_CHECKS=0');

Delete all records in specific tables.

DB::table('users')->truncate();

DB::table('posts')->truncate();

DB::table('roles')->truncate();

DB::table('categories')->truncate();

DB::table('photos')->truncate();

DB::table('comments')->truncate();

DB::table('comment\_replies')->truncate();

Run the Users factory creating 10 records:

factory(App\User::class, 10)->create()->each(function ($user) {

$user->posts()->save(factory(App\Post::class)->make());

});

Run the Roles factory creating 3 records:

factory(App\Role::class, 3)->create();

If you try to create more than 3 records the factory will throw an error due to the use of unique() in the factory.

Run the Categories factory creating 10 records:

factory(App\Category::class, 10)->create();

If you try to create more than 10 records the factory will throw an error due to the use of unique() in the factory.

Run the Photos factory creating 10 records:

factory(App\Photo::class, 1)->create();

Run the Comments factory creating 10 Comment records and 10 Comment\_Replies records:

factory(App\Comment::class, 10)->create()->each(function ($c) {

$c->replies()->save(factory(App\CommentReply::class)->make());

});

It may not be one Comment to one Post but it is one Comment\_Reply to one Comment.

# Section 40: Upgrade to Laravel 5.5

## Warning PHP Update for 5.5 - (Edwin from the future)

You required to upgrade too PHP 7.x before upgrading to Laravel 5.5

This covers the upgrade of PHP across various Windows, Mac, Vagrant and XAMPP.

## Laravel Homestead Box Update (Edwin from the future)

This is for those Using Vagrant / Homestead - Everyone else disregard this lecture

1. Vagrant box list - this will show the version number and boxes - write down or copy this information
2. Vagrant box update - downloads the and install the updates
3. Vagrant destroy - destroy the old box
4. Vagrant up - This provisions the box and now the changes will take effect

You can purge the old box by doing:

Vagrant box remove laravel/homestead --box-version=5.1.0

--------------

This command could change so make sure you check the Documentations all the time!

## Upgrading the easy way

For more information see: <https://laravel.com/docs/5.5/upgrade#upgrade-5.5.0>

Modify the file composer.json. See changes in ***bold***.

. . .

. . .

"php": ">=5.5.9",

***"laravel/framework": "5.5.\*",***

***"laravelcollective/html": "5.5.\*",***

***"cviebrock/eloquent-sluggable": "^4.3",***

***"filp/whoops": "^2.5",***

. . .

. . .

And

. . .

. . .

"mockery/mockery": "0.9.\*",

***"phpunit/phpunit": "~6.0",***

"symfony/css-selector": "2.8.\*|3.0.\*",

. . .

. . .

And

. . .

. . .

"post-update-cmd": [

"Illuminate\\Foundation\\ComposerScripts::postUpdate",

"php artisan optimize"

]***,***

***"post-autoload-dump": [***

***"Illuminate\\Foundation\\ComposerScripts::postAutoloadDump",***

***"@php artisan package:discover"***

***]***

. . .

. . .

Make sure Composer is up to date by running:

composer self-update

Run:

composer update

Now run the Laravel and support package update with:

composer install

Run:

npm install

Flush the browser and Laravel view and route cache:

php artisan view:clear

php artisan route:clear

Double check the version of Laravel (should be 5.5.x):

php artisan --version

Generate the APP\_KEY:

php artisan key:generate

Login and test the application.

# Section 41: Upgrade to Laravel 5.6

## Upgrading to Laravel 5.6

For more information see: <https://laravel.com/docs/5.6/upgrade#upgrade-5.6.0>

You required to upgrade to PHP 7.1.3 or higher before upgrading to Laravel 5.5

Discussed the php upgrade across various platforms, Windows Mac, Ubuntu, Homestead.

Free course on Laravel Homestead in Udemy: <https://www.udemy.com/course/laravel-homestead/>

Modify composer.json.

Remove all instances of: "php artisan optimize"

Make the changes in ***bold***.

"require": {

***"php": ">=7.0.0",***

***"laravel/framework": "5.6.\*",***

"laravelcollective/html": "5.5.\*",

***"cviebrock/eloquent-sluggable": "^4.5",***

Run:

Composer update

Run:

Composer install

Flush the browser and Laravel view and route cache:

php artisan view:clear

php artisan route:clear

Double check the version of Laravel (should be 5.6.x):

php artisan --version

Login and test the application.

# Section 42: Extra - Adjustments and more Features

## Making admin more modular

Creating three includes from layouts.admin.

admin\_nav.blade.php

Contains the division:

<nav

class="navbar navbar-default navbar-static-top"

role="navigation"

style="margin-bottom: 0"

>

Included in: layouts.admin

admin\_side\_nav.blade.php

Contains the division:

<div class="navbar-default sidebar" role="navigation">

Included in: layouts.admin\_nav

admin\_top\_nav.blade.php

Contains the division:

<div class="navbar-header">

And

<ul class="nav navbar-top-links navbar-right">

Included in: layouts.admin\_nav

## Creating an admin controller

Created the controller AdminController.

php artisan make:controller AdminController

Created the index method in the controller AdminController.

public function index()

{

return view('admin/index');

}

Modified the ‘/admin’ route in web.php to access the index method of the AdminController.

Route::get('/admin', 'AdminController@index');

Moved the home view to the new front directory under views.

Created the index method in the controller HomeController to display the home view.

public function index()

{

return view('front/home');

}

Better way to manage admin routes?????

Route::middleware(['admin'])->prefix('admin')->name('admin.')->group(function() {

Route::resource('users', 'AdminUsersController');

Route::resource('/posts', 'AdminPostsController');

Route::resource('/categories', 'AdminCategoriesController');

Route::resource('/media', 'AdminMediaController');

Route::post('/media/delete',

'AdminMediaController@deleteMedia')->name('media.delete');

Route::resource('/comments', 'PostCommentController');

Route::resource('/comments/replies', 'CommentRepliesController');

Route::get('/', 'AdminController@index')->name('dashboard');

});

## Installing a Chart on admin

Using Chart.js v.2.8

Website: <https://www.chartjs.org/>

Modify the view admin.index.

@extends('layouts.admin');

@section('content')

<h1>Admin</h1>

<canvas id="myChart"></canvas>

<hr>

@endsection

@section('scripts')

<script src=<https://cdn.jsdelivr.net/npm/chart.js@2.8.0/dist/Chart.min.js>

integrity="sha256-Uv9BNBucvCPipKQ2NS9wYpJmi8DTOEfTA/nH2aoJALw="

crossorigin="anonymous">

</script>

<script>

var ctx = document.getElementById('myChart').getContext('2d');

var myChart = new Chart(ctx, {

type: 'bar',

data: {

labels: ['Posts', 'Categories', 'Comments'],

datasets: [{

label: 'Data of CMS',

data: [12, 19, 3, 5, 2, 3],

backgroundColor: [

'rgba(255, 99, 132, 0.2)',

'rgba(54, 162, 235, 0.2)',

'rgba(255, 206, 86, 0.2)',

'rgba(75, 192, 192, 0.2)',

'rgba(153, 102, 255, 0.2)',

'rgba(255, 159, 64, 0.2)'

],

borderColor: [

'rgba(255, 99, 132, 1)',

'rgba(54, 162, 235, 1)',

'rgba(255, 206, 86, 1)',

'rgba(75, 192, 192, 1)',

'rgba(153, 102, 255, 1)',

'rgba(255, 159, 64, 1)'

],

borderWidth: 1

}]

},

options: {

scales: {

yAxes: [{

ticks: {

beginAtZero: true

}

}]

}

}

});

</script>

@endsection

## Replacing Chart static data with dynamic data

Modify the index method of the controller AdminController.

public function index()

{

$postsCount = Post::count();

$categoriesCount = Category::count();

$commentsCount = Comment::count();

return view(

'admin/index',

compact('postsCount', 'categoriesCount', 'commentsCount')

);

}

Modify the view admin.index. See changes in ***bold***.

datasets: [{

. . .

. . .

data: [

{{ $postsCount }},

{{ $categoriesCount }},

{{ $commentsCount }}

],

. . .

. . .

}]

## Some adjustments to admin posts page

Modified (cleaned up) the view admin.posts.index. See changes in ***bold***.

<thead>

<tr>

<th>Id</th>

<th>Photo</th>

<th>Title</th>

<th>Owner</th>

<th>Category</th>

<th></th>

<th></th>

<th>Created</th>

<th>Updated</th>

</tr>

</thead>

<tbody>

@if($posts)

@foreach($posts as $post)

<tr>

<td>{{$post->id}}</td>

<td><img height="50" src="{{$post->Photo ? $post->Photo->file : 'https://via.placeholder.com/50'}}" alt="Not Avail."></td>

<td>{{$post->title}}</td>

<td>{{$post->user->name}}</td>

<td>{{$post->category ? $post->category->name : "Uncategorized" }}</td>

<td><a href="{{ route('home.post', $post->slug) }}">View Post</a></td>

<td><a href="{{ route('admin.comments.show', $post->id) }}">

View Comments

</a></td>

<td>{{$post->created\_at->diffForHumans()}}</td>

<td>{{$post->updated\_at->diffForHumans()}}</td>

</tr>

@endforeach

@endif

</tbody>

## Installing a Blog Home Page

Modify the view front.home.

@extends('layouts.blog-home')

Modify the view layouts.blog-home.

<!-- Navigation -->

@include('includes.front\_nav')

<!-- Blog Sidebar Widgets Column -->

@include('includes.front\_sidebar')

Create the view includes.front\_nav.

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header">

<button

type="button"

class="navbar-toggle"

data-toggle="collapse"

data-target="#bs-example-navbar-collapse-1"

>

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">Start Bootstrap</a>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div

class="collapse navbar-collapse"

id="bs-example-navbar-collapse-1"

>

<ul class="nav navbar-nav">

<li>

<a href="#">About</a>

</li>

<li>

<a href="#">Services</a>

</li>

<li>

<a href="#">Contact</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container -->

</nav>

Create the view includes.front\_sidebar.

<div class="col-md-4">

<!-- Blog Search Well -->

<div class="well">

<h4>Blog Search</h4>

<div class="input-group">

<input type="text" class="form-control">

<span class="input-group-btn">

<button class="btn btn-default" type="button">

<span class="glyphicon glyphicon-search"></span>

</button>

</span>

</div>

<!-- /.input-group -->

</div>

<!-- Blog Categories Well -->

<div class="well">

<h4>Blog Categories</h4>

<div class="row">

<div class="col-lg-6">

<ul class="list-unstyled">

<li><a href="#">Category Name</a>

</li>

<li><a href="#">Category Name</a>

</li>

<li><a href="#">Category Name</a>

</li>

<li><a href="#">Category Name</a>

</li>

</ul>

</div>

<!-- /.col-lg-6 -->

<div class="col-lg-6">

<ul class="list-unstyled">

<li><a href="#">Category Name</a>

</li>

<li><a href="#">Category Name</a>

</li>

<li><a href="#">Category Name</a>

</li>

<li><a href="#">Category Name</a>

</li>

</ul>

</div>

<!-- /.col-lg-6 -->

</div>

<!-- /.row -->

</div>

<!-- Side Widget Well -->

<div class="well">

<h4>Side Widget Well</h4>

<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Inventore,

perspiciatis adipisci accusamus laudantium odit aliquam repellat

tempore quos aspernatur vero.

</p>

</div>

</div>

## Making front-end more modular

Modified the view layouts.blog-home.blade.php.

Moved <head> section to includes.front\_header.blade.php.

Moved <footer> and closing tags to includes.front\_footer.blade.php.

Moved <div class=”container”> tag to includes.front\_nav.blade.php.

Leaving:

@include('includes.front\_header')

<!-- Navigation -->

@include('includes.front\_nav')

<div class="row">

<!-- Blog Entries Column -->

<div class="col-md-8">

<!-- First Blog Post -->

<h2>

<a href="#">Blog Post Title</a>

</h2>

<p class="lead">

by <a href="index.php">Start Bootstrap</a>

</p>

<p>

<span class="glyphicon glyphicon-time"></span>

Posted on August 28, 2013 at 10:00 PM

</p>

<hr>

<img

class="img-responsive"

src="http://placehold.it/900x300" alt=""

>

<hr>

<p>

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Dolore, veritatis, tempora, necessitatibus inventore nisi

quam quia repellat ut tempore laborum possimus eum dicta id

animi corrupti debitis ipsum officiis rerum.

</p>

<a class="btn btn-primary" href="#">

Read More

<span class="glyphicon glyphicon-chevron-right"></span>

</a>

<hr>

<!-- Pagination -->

</div>

<!-- Blog Sidebar Widgets Column -->

@include('includes.front\_sidebar')

</div>

<!-- /.row -->

<hr>

@include('includes.front\_footer')

Created the view includes.front\_header.blade.php.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="description" content="">

<meta name="author" content="">

<title>Blog Home - Start Bootstrap Template</title>

<!-- Bootstrap Core CSS -->

<link rel="stylesheet" href="{{ asset('css/libs.css') }}">

<!-- HTML5 Shim and Respond.js IE8 support of HTML5 elements and media

queries -->

<!-- WARNING: Respond.js doesn't work if you view the page via file:// -->

<!--[if lt IE 9]>

<script

src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js">

</script>

<script

src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js">

</script>

<![endif]-->

</head>

<body>

Created the view includes.front\_footer.blade.php.

<!-- Footer -->

<footer>

<div class="row">

<div class="col-lg-12">

<p>Copyright &copy; Your Website 2014</p>

</div>

<!-- /.col-lg-12 -->

</div>

<!-- /.row -->

</footer>

</div>

<!-- /.container -->

<!-- jQuery -->

<script src="js/jquery.js"></script>

<!-- Bootstrap Core JavaScript -->

<script src="js/bootstrap.min.js"></script>

</body>

</html>

Modified the view includes.front\_nav.blade.php.

Addded the following at the bottom:

<!-- Page Content -->

<div class="container">

This was done to balance out the opening can closing tags between front\_nav, blog-home and front\_footer.

## Replacing static with dynamic data for home page

Problem: Laravel 5.8 no longer supporting Laravel Html Collective

Answer: make sure to use this version of Laravel collective:

"laravelcollective/html": "5.8.\*",

(paste that inside require section in composer.json and then type: composer install)

Modify the index method of the controller HomeController.

public function index()

{

$posts = Post::paginate(2);

return view('front/home', compact('posts'));

}

Modify the view layouts.blog-home.

The Page Content section (<div class="row">) get replaced with:

<!-- Page Content -->

@yield('content')

Modify the view front.home. See changes in ***bold***.

@extends('layouts.blog-home')

@section('content')

<div class="row">

<!-- Blog Entries Column -->

<div class="col-md-8">

***@if($posts)***

***@foreach ($posts as $post)***

<!-- First Blog Post -->

<h2>

<a href="#">***{{ $post->title }}***</a>

</h2>

<p class="lead">

by <a href="index.php">***{{ $post->user->name }}***</a>

</p>

<p>

<span class="glyphicon glyphicon-time"></span>

Posted on ***{{ $post->created\_at->diffForHumans() }}***

</p>

<hr>

<img

class="img-responsive"

src="http://placehold.it/900x300" alt=""

>

<hr>

<p>***{{ str\_limit($post->body,300) }}***</p>

<a

class="btn btn-primary"

href="***/codehacking/public/post/{{ $post->slug }}***"

>

Read More

<span class="glyphicon glyphicon-chevron-right"></span>

</a>

<hr>

<!-- Pagination -->

***@endforeach***

***@else***

***@endif***

</div>

<!-- Blog Sidebar Widgets Column -->

@include('includes.front\_sidebar')

</div>

<!-- /.row -->

<hr>

@endsection

## Some adjustments to single post page

Updated the views includes.front\_sidebar to display a categories list.

<ul class="list-unstyled">

@if($categories)

@foreach ($categories as $category)

<li><a href="#">{{ $category->name }}</a></li>

@endforeach

@endif

</ul>

Modified the index method of the controller HomeController to generate the categories list.

public function index()

{

$posts = Post::paginate(2);

$categories = Category::orderBy('name')->get();

return view('front/home', compact('posts', 'categories'));

}

Modified the view post to extend the view layouts.blog-home instead of layouts.blog-post.

Originally: @extends('layouts.blog-post')

Now: @extends('layouts.blog-home')

Modified the store method of the controller PostCommentsController to correct the error generated by a missing photo.

$photoName = $user->photo ?

substr(strstr($user->photo->file, '/images/'), 8):

"";

## Making front-end top navigation better

Modified the view includes.front\_nav.

Turned Codehacking into a link:

<a class="navbar-brand" href="{{ url('/') }}">Codehacking</a>

Shifted the nav menu to the right and changed the menu options. See changes in ***bold***.

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav ***navbar-right***">

***@if(Auth::guest())***

***<li>***

***<a href="{{ url('/login') }}">Login</a>***

***</li>***

***<li>***

***<a href="{{ url('/register') }}">Register</a>***

***</li>***

***@else***

***<li>***

***<a href="{{ url('/admin') }}">Admin</a>***

***</li>***

***<li>***

***<a href="{{ url('/logout') }}">Logout</a>***

***</li>***

***@endif***

</ul>

</div>

Modified the view Auth.login to use the blog-home layout as its layout. To give it the proper top nav menu.

Before: @extends('layouts.app')

After: @extends('layouts.blog-home')

Modified the view includes.front\_footer to display the correct company and copyright year.

<p>Copyright &copy; Codehacking.test {{ \Carbon\Carbon::now()->year }}</p>

Modified the view front.home to remove the link from the author’s name.

by {{ $post->user->name }}

## Front-end Comment Flash messaging modification

Moved display of session messages into its own view at includes.flash\_messages.

Modified view post.

Move display of session me to the view includes.flash\_messages.

Contained post/comment content into an 8 column section on the left.

Restored the view includes.front\_sidebar into a 4 column section on the right.

Modified the view layouts.blog-home to display the session messages with includes.flash\_messages.

## Pagination in the front-end

Implemented pagination at the bottom of the home page ([http://tutorial.tst/codehacking/public](http://tutorial.tst/codehacking/public/?page=1)). I did that already.

Change language in Laravel:

<https://laracasts.com/discuss/channels/eloquent/change-the-language-of-carbon>

<https://laracasts.com/discuss/channels/general-discussion/changing-language-to-carbon>

Basically you can go to AppServiceProvider and in boot() method try to set the language:

public function boot() { Carbon::setLocale('hr');

Steps to do (for other people):

1. Go to App->Providers->AppServiceProviders.php add Carbon::setLocale('de'); to the public function boot() ['de' is the language, if you need another one go look for supported languages]
2. Add 'Carbon' => 'Carbon\Carbon', on the bottom of your aliases in config->app.php
3. Voilà it works!

# Section 43: Upgrade from 5.6 to 5.7 & 5.7 to 5.8

## Let's start upgrading to 5.7

Modify composer.json. See changes in ***bold***.

"require": {

***"php": ">=7.1.3",***

***"laravel/framework": "5.7.\*",***

"laravelcollective/html": "5.5.\*",

"cviebrock/eloquent-sluggable": "^4.5",

"filp/whoops": "^2.5",

"unisharp/laravel-filemanager": "^1.6",

"intervention/image": "^2.3"

},

"require-dev": {

"fzaninotto/faker": "~1.4",

"mockery/mockery": "0.9.\*",

"phpunit/phpunit": "~6.0", ***<<-- Remove!!!***

"symfony/css-selector": "2.8.\*|3.0.\*",

"symfony/dom-crawler": "2.8.\*|3.0.\*"

},

Modify the file {application root}\config\app.php. Add the following to the providers array property:

/\*

\* added dur to laravel 5.7 upgrade

\*/

Illuminate\Notifications\NotificationServiceProvider::class,

Run: composer update

For more information see: <https://laravel.com/docs/5.7/upgrade>

## Let's start upgrading to 5.8

Modify composer.json. See changes in ***bold***.

"require": {

"php": ">=7.1.3",

***"laravel/framework": "5.8.\*",***

"laravelcollective/html": "\*",

"cviebrock/eloquent-sluggable": "^4.5",

"filp/whoops": "^2.5",

"unisharp/laravel-filemanager": "^1.6",

"intervention/image": "^2.3"

},

Run: composer update

For more information see: <https://laravel.com/docs/5.8/upgrade>

# Appendix

## Certificate of Compleation



## Special Note – Compatibility Issues

Lavavel v.5.2 has issues working with PHP 7.2 and higher. Only use PHP 7.1.x and lower when using Laravel 5.2.

## Composer – Installing Laravel

Composer create-project laravel/laravel ***one2one*** 5.2.31 --prefer-dist

***one2one*** – The name of the directory where the installation should land.

To install the latest version just omit the version like this:

composer create-project --prefer-dist laravel/laravel cms

## Listing Routes

php artisan route:list

## Creating Controllers

php artisan make:controller PostController

php artisan make:controller --resource PostController

## Creating Migrations

Command to create a migration to ***create*** a table:

php artisan make:migration create\_posts\_table --create=”posts”

Command to create a migration to ***modify*** a table:

php artisan make:migration add\_slug\_to\_posts\_table --table=”posts”

Double quotes(“) on posts are optional

If you’re adding and dropping tables use the method Schema::create or Schema::drop

If you’re modifying a table (add/drop columns) use the method Schema::table

## Creating Requests

php artisan make:request {name of request}

Form Request Validation. For more complex validation scenarios, you may wish to create a "form request". Form requests are custom request classes that contain validation logic. The generated class will be placed in the {application root}app/Http/Requests directory.

## Implementing Migrations

php artisan migrate Apply migrations

php artisan migrate:rollback Remove last applied migration

php artisan migrate:reset Remove all migrations

php artisan migrate:refresh Does a reset and a migrate in one shot

php artisan migrate:status Gives a listing of what migrations have been applied

## Creating Models

php artisan make:model Post

php artisan make:model Role –m Creating both the model and migration.

## Creating Login in Under 1 Minute

php artisan make:auth Create login views and controllers that work with provided user and password models and migrations.

## Tinker

php artisan tinker Enter Tinker

## File Locations

{application root}\database\migrations Directory of database migration programs

{application root}\app\ Directory of data model programs

{application root}\app\http\routes.php Routes file

{application root}\app\http\controllers Directory of controller files

{application root}\resources\views Directory of view files

{application root}\Http\Requests Directory of validation request objects

## Create Middleware

To create a skeleton for middleware user the following;

php artisan make:middleware RoleMiddleware

## GIT Commands

git init

git status

git add {file skeleton}

git log

git commit -m "{description of commit}"

git reset –hard {reference}

git checkout –b newcategory

git checkout newcategory

Git merge newcategory

git checkout –d newcategory

git config --global user.name "{your name}" Set your name in GIT

git config --global user.email "{valid email}" Set your email address in GIT

git clone {url provided by the repository}

git push origin master

git remote add origin <https://github.com/brayfiel/codehacking.git>

git pull

git show --pretty="" --name-only {Git log change reference}

## Webpack

Run the following to resolve a missing dependency:

npm install --save-dev node-sass@4.12

Run the following to compile the change to app.scss:

npm run dev

## Laravel Collective HTML Docs

From: <https://github.com/LaravelCollective/docs/blob/5.2/html.md>

### Forms & HTML

* [Installation](https://github.com/LaravelCollective/docs/blob/5.2/html.md#installation)
* [Opening A Form](https://github.com/LaravelCollective/docs/blob/5.2/html.md#opening-a-form)
* [Form Model Binding](https://github.com/LaravelCollective/docs/blob/5.2/html.md#form-model-binding)
* [Form Model Accessors](https://github.com/LaravelCollective/docs/blob/5.2/html.md#form-model-accessors)
* [CSRF Protection](https://github.com/LaravelCollective/docs/blob/5.2/html.md#csrf-protection)
* [Labels](https://github.com/LaravelCollective/docs/blob/5.2/html.md#labels)
* [Text, Text Area, Password & Hidden Fields](https://github.com/LaravelCollective/docs/blob/5.2/html.md#text)
* [Checkboxes and Radio Buttons](https://github.com/LaravelCollective/docs/blob/5.2/html.md#checkboxes-and-radio-buttons)
* [File Input](https://github.com/LaravelCollective/docs/blob/5.2/html.md#file-input)
* [Number Input](https://github.com/LaravelCollective/docs/blob/5.2/html.md#number)
* [Date Input](https://github.com/LaravelCollective/docs/blob/5.2/html.md#date)
* [Drop-Down Lists](https://github.com/LaravelCollective/docs/blob/5.2/html.md#drop-down-lists)
* [Buttons](https://github.com/LaravelCollective/docs/blob/5.2/html.md#buttons)
* [Custom Macros](https://github.com/LaravelCollective/docs/blob/5.2/html.md#custom-macros)
* [Custom Components](https://github.com/LaravelCollective/docs/blob/5.2/html.md#custom-components)
* [Generating URLs](https://github.com/LaravelCollective/docs/blob/5.2/html.md#generating-urls)

### Installation

Begin by installing this package through Composer. Run the following from the Terminal:

composer require "laravelcollective/html":"^5.2.0"

Next, add your new provider to the providers array of config/app.php:

'providers' => [

// ...

Collective\Html\HtmlServiceProvider::class,

// ...

],

Finally, add two class aliases to the aliases array of config/app.php:

'aliases' => [

// ...

'Form' => Collective\Html\FormFacade::class,

'Html' => Collective\Html\HtmlFacade::class,

// ...

],

Looking to install this package in [Lumen](http://lumen.laravel.com/)? First of all, making this package compatible with Lumen will require some core changes to Lumen, which we believe would dampen the effectiveness of having Lumen in the first place. Secondly, it is our belief that if you need this package in your application, then you should be using Laravel anyway.

### Opening A Form

{!! Form::open(['url' => 'foo/bar']) !!}

//

{!! Form::close() !!}

By default, a POST method will be assumed; however, you are free to specify another method:

echo Form::open(['url' => 'foo/bar', 'method' => 'put'])

**Note:** Since HTML forms only support POST and GET, PUT and DELETE methods will be spoofed by automatically adding a \_method hidden field to your form.

You may also open forms that point to named routes or controller actions:

echo Form::open(['route' => 'route.name'])

echo Form::open(['action' => 'Controller@method'])

You may pass in route parameters as well:

echo Form::open(['route' => ['route.name', $user]])

echo Form::open(['action' => ['Controller@method', $user]])

If your form is going to accept file uploads, add a files option to your array:

echo Form::open(['url' => 'foo/bar', 'files' => true])

### Form Model Binding

#### Opening A Model Form

Often, you will want to populate a form based on the contents of a model. To do so, use the Form::model method:

echo Form::model($user, ['route' => ['user.update', $user]])

Now, when you generate a form element, like a text input, the model's value matching the field's name will automatically be set as the field value. So, for example, for a text input named email, the user model's email attribute would be set as the value. However, there's more! If there is an item in the Session flash data matching the input name, that will take precedence over the model's value. So, the priority looks like this:

1. Session Flash Data (Old Input)
2. Explicitly Passed Value
3. Model Attribute Data

This allows you to quickly build forms that not only bind to model values, but easily re-populate if there is a validation error on the server!

**Note:** When using Form::model, be sure to close your form with Form::close!

#### Form Model Accessors

Laravel's [Eloquent Accessor](http://laravel.com/docs/5.2/eloquent-mutators#accessors-and-mutators) allow you to manipulate a model attribute before returning it. This can be extremely useful for defining global date formats, for example. However, the date format used for display might not match the date format used for form elements. You can solve this by creating two separate accessors: a standard accessor, and/or a form accessor.

To use form accessors, first include the FormAccessible trait in the model, then create a formFooAttribute method on your model where Foo is the "camel" cased name of the column you wish to access. In this example, we'll define an accessor for the date\_of\_birth attribute. The accessor will automatically be called by the HTML Form Builder when attempting to pre-fill a form field when Form::model() is used.

<?php

namespace App;

use Carbon\Carbon;

use Illuminate\Database\Eloquent\Model;

use Collective\Html\Eloquent\FormAccessible;

class User extends Model

{

use FormAccessible;

/\*\*

\* Get the user's first name.

\*

\* @param string $value

\* @return string

\*/

public function getDateOfBirthAttribute($value)

{

return Carbon::parse($value)->format('m/d/Y');

}

/\*\*

\* Get the user's first name for forms.

\*

\* @param string $value

\* @return string

\*/

public function formDateOfBirthAttribute($value)

{

return Carbon::parse($value)->format('Y-m-d');

}

}

### CSRF Protection

If you use the Form::open or Form::model method with POST, PUT or DELETE the CSRF token used by Laravel for CSRF protection will be added to your forms as a hidden field automatically. Alternatively, if you wish to generate the HTML for the hidden CSRF field, you may use the token method:

echo Form::token();

For more information on Laravel's CSRF protection, see [the relevant section in Laravel's documentation](https://laravel.com/docs/5.2/routing#csrf-protection).

### Labels

#### Generating A Label Element

echo Form::label('email', 'E-Mail Address');

#### Specifying Extra HTML Attributes

echo Form::label('email', 'E-Mail Address', ['class' => 'awesome']);

**Note:** After creating a label, any form element you create with a name matching the label name will automatically receive an ID matching the label name as well.

### Text, Text Area, Password & Hidden Fields

#### Generating A Text Input

echo Form::text('username');

#### Specifying A Default Value

echo Form::text('email', 'example@gmail.com');

**Note:** The hidden and textarea methods have the same signature as the text method.

#### Generating A Password Input

echo Form::password('password', ['class' => 'awesome']);

#### Generating Other Inputs

echo Form::email($name, $value = null, $attributes = array());

echo Form::file($name, $attributes = array());

### Checkboxes and Radio Buttons

#### Generating A Checkbox Or Radio Input

echo Form::checkbox('name', 'value');

echo Form::radio('name', 'value');

#### Generating A Checkbox Or Radio Input That Is Checked

echo Form::checkbox('name', 'value', true);

echo Form::radio('name', 'value', true);

### Number

#### Generating A Number Input

echo Form::number('name', 'value');

### Date

#### Generating A Date Input

echo Form::date('name', \Carbon\Carbon::now());

### File Input

#### Generating A File Input

echo Form::file('image');

**Note:** The form must have been opened with the files option set to true.

### Drop-Down Lists

#### Generating A Drop-Down List

echo Form::select('size', ['L' => 'Large', 'S' => 'Small']);

#### Generating A Drop-Down List With Selected Default

echo Form::select('size', ['L' => 'Large', 'S' => 'Small'], 'S');

#### Generating a Drop-Down List With an Empty Placeholder

This will create an <option> element with no value as the very first option of your drop-down.

echo Form::select('size', ['L' => 'Large', 'S' => 'Small'], null, ['placeholder' => 'Pick a size...']);

#### Generating a List With Multiple Selectable Options

echo Form::select('size', array('L' => 'Large', 'S' => 'Small'), null, array('multiple' => true));

#### Generating A Grouped List

echo Form::select('animal', [

'Cats' => ['leopard' => 'Leopard'],

'Dogs' => ['spaniel' => 'Spaniel'],

]);

#### Generating A Drop-Down List With A Range

echo Form::selectRange('number', 10, 20);

#### Generating A List With Month Names

echo Form::selectMonth('month');

### Buttons

#### Generating A Submit Button

echo Form::submit('Click Me!');

**Note:** Need to create a button element? Try the button method. It has the same signature as submit.

### Custom Macros

#### Registering A Form Macro

It's easy to define your own custom Form class helpers called "macros". Here's how it works. First, simply register the macro with a given name and a Closure:

Form::macro('myField', function()

{

return '<input type="awesome">';

});

Now you can call your macro using its name:

#### Calling A Custom Form Macro

echo Form::myField();

##Custom Components

#### Registering A Custom Component

Custom Components are similar to Custom Macros, however instead of using a closure to generate the resulting HTML, Components utilize [Laravel Blade Templates](http://laravel.com/docs/5.2/blade). Components can be incredibly useful for developers who use [Twitter Bootstrap](http://getbootstrap.com/), or any other front-end framework, which requires additional markup to properly render forms.

Let's build a Form Component for a simple Bootstrap text input. You might consider registering your Components inside a Service Provider's boot method.

Form::component('bsText', 'components.form.text', ['name', 'value', 'attributes']);

Notice how we reference a view path of components.form.text. Also, the array we provided is a sort of method signature for your Component. This defines the names of the variables that will be passed to your view. Your view might look something like this:

// resources/views/components/form/text.blade.php

<div class="form-group">

{{ Form::label($name, null, ['class' => 'control-label']) }}

{{ Form::text($name, $value, array\_merge(['class' => 'form-control'], $attributes)) }}

</div>

Custom Components can also be created on the Html facade in the same fashion as on the Form facade.

##### Providing Default Values

When defining your Custom Component's method signature, you can provide default values simply by giving your array items values, like so:

Form::component('bsText', 'components.form.text', ['name', 'value' => null, 'attributes' => []]);

#### Calling A Custom Form Component

Using our example from above (specifically, the one with default values provided), you can call your Custom Component like so:

{{ Form::bsText('first\_name') }}

This would result in something like the following HTML output:

<div class="form-group">

<label for="first\_name">First Name</label>

<input type="text" name="first\_name" value="" class="form-control">

</div>

##Generating URLs

#### link\_to

Generate a HTML link to the given URL.

echo link\_to('foo/bar', $title = null, $attributes = array(), $secure = null);

#### link\_to\_asset

Generate a HTML link to the given asset.

echo link\_to\_asset('foo/bar.zip', $title = null, $attributes = array(), $secure = null);

#### link\_to\_route

Generate a HTML link to the given named route.

echo link\_to\_route('route.name', $title = null, $parameters = array(), $attributes = array());

#### link\_to\_action

Generate a HTML link to the given controller action.

echo link\_to\_action('HomeController@getIndex', $title = null, $parameters = array(), $attributes = array());